Semestrální práce z KIV/PRO

Hledání všech k nejbližších sousedů

Jiří Velek

A20B0269P

jvelek@students.zcu.cz

30. 10. 2021

Zadání

Najděte v anglicky psané odborné literatuře článek v délce alespoň 5 stránek o nějakém algoritmu řešícím libovolný problém. Algoritmus popište do českého referátu tak, aby ho podle vašeho názoru pochopil běžný student 2. ročníku informatiky. Váš text musí svědčit o tom, že algoritmu rozumíte, a musí ho z něj pochopit i nezasvěcený čtenář.

Úvod

Existuje mnoho algoritmů pro hledání k nejbližších sousedů, plno z nich využívají techniky jako je rozděl a panuj, nebo Voronoi diagramy [1, 2, 3]. Náš algoritmus pracuje bez složitého předzpracování.

Popis a definice problému

Problém je nalezení algoritmu, který k množině n bodů $P[x_i, y_i]$, $i = 1, \ldots, n$ a množině indexů $j_1, \ldots, m$, $1 \leq j_k \leq n$ najde k množině $P_{j_1}, \ldots, P_{j_m}$ k nejbližších sousedů.

Popis algoritmu

Nejprve je potřeba si připravit uniformní mřížku, do které jsou vloženy všechny body. Mřížka je navržena tak, aby se do jedné buňky vešlo více bodů. Požadavky na mřížku jsou následující:

- mřížku lze vytvořit v lineárním čase
- výkon se výrazně neliší v závislosti na tom, jestli jsou data uniformně rozložena nebo ne
- velikost mřížky se dá jednoduše upravit pro jakákoliv praktická data

Všechny body jsou vloženy do mřížky tak, aby byly zachovány vzdálenosti mezi jednotlivými body.

Vyhledávání sousedů probíhá po vrstvách, začne se od bodu P_{j_r} a postupuje se "v kružnici" od daného bodu. Vyhledávání skončí, když k-tý nejbližší bod má menší vzdálenost, než vzdálenost kandidáta P a nejbližší stěny buňky,

ve které P leží plus poloměr kružnice, ve které se momentálně algoritmus nachází.

Při vyhledávání se vzdálenosti bodů uchovávají v seřazeném poli, když algoritmus najde bod s kratší vzdáleností, než poslední prvek v tomto poli, nahradí ho nalezenou vzdáleností a znovu pole seřadí.

Po dokončení hlavní smyčky algoritmu obsahuje pole vzdáleností k nejbližších sousedů. Indexy bodů ke kterým tyto vzdálenosti patří jsou daní sousedé.

Sestrojení uniformní mřížky

Mřížku je třeba sestrojit tak, aby její jednotlivé buňky neobsahovali příliš mnoho bodů. Algoritmus najde maximální a minimální x a y souřadnice bodu (x_{max}, x_{min}) a (y_{max}, y_{min})

Jedna buňka má potom velikost:

$$velikost = \alpha\sqrt{\frac{(x_{max} - x_{min})(y_{max} - y_{min})}{n}}$$

, kde parametr α umožňuje upravit měřítko mřížky. Šířku a výšku mřížky lze spočítat jako:

$$xres = \left\lfloor \frac{x_{max} - x_{min}}{velikost} \right\rfloor$$
$$yres = \left\lfloor \frac{y_{max} - y_{min}}{velikost} \right\rfloor$$

Pro vložení bodů do mřížky je potřeba vypočítat pro každý bod $P[x, y]$ jeho souřadnice buňky i, j

$$i = \left\lfloor \frac{x - x_{min}}{xres} \right\rfloor$$
$$j = \left\lfloor \frac{y - y_{min}}{yres} \right\rfloor$$

Pseudokód

vstup: množina bodů P, množina indexů j

výstup: dvourozměrné pole, $c[m][k]$ s k nejbližšími sousedy

```
for r = 1 to m by 1:
  (ic, jc) = indexy v mřížce bodu Pjr
  dsh = vzdálenost od bodu Pjr k nejbližší stěne buňky (ic, jc)

  for i = 1 to k by 1:
    bucket[i] = infinity
  dmin = infinity
  l = 0

  while dmin > dsh:
    il = max(1, ic - l)
    jl = max(1, jc - l)
    ih = min(xres, ic + l)
    jh = min(yres, jc + l)

    for i = il to ih by 1:
      if i == il or i == ih:
        ji = 1
      else:
        ji = jh - jl

      for j = jl to jh by ji:
        if cell[i][j] ještě nebyla navštívena a není prázdná:
          dis = vzdálenost bodu Pjr a bodů v cell[i][j]

          if dis < bucket[k]:
            bucket[k] = dis
            ind[k] = index nalezeného bodu
            seřadit bucket a ind

          označit cell[i][j] jako navštívena

    dmin = bucket[k]
    dsh = dsh + velikost
    l = l + 1

  for s = 1 to k by 1:
    c[r][s] = ind[s]
```

Složitost algoritmu

Algoritmus byl otestován na několika různých datech. Data set č. 1 obsahuje nejjednodušší případy, kdy jsou body uniformně rozděleny. Data set č. 2 obsahuje velké mezery mezi body, a díru ve středu. Data set č. 3a obsahuje případ, kdy se sousedé hledají pro izolovaný bod. Data set č. 3b jsou stejná jako data set 3a s jedinou vyjímkou - sousedé se vyhledávají pro bod mimo díru. Data set č. 4 obsahuje všechny body ve dvou "množinách" a vyhledávání probíhá v jedné z nich. V data setech na obrázcích bylo použito $n = 1000$ a $k = 50$

Tabulka 1 popisuje zaplnění uniformní mřížky

Data set č.	1	2	3(a, b)	4
využitých buněk	62.8%	39.6%	51.6%	28.9%

Tabulka 1: Zaplnění buněk v mřížce [4]

Pro data č. 1 a 2 má algoritmus lineární složitost, i přesto, že mřížka v druhém případě obsahuje 60.4% nevyužitých buněk.

Pokud je bod, pro který jsou sousedé hledány, umístěn v izolaci (jako v případě č. 3a), časová složitost algoritmu se zhorší. Při vybrání jiného bodu (případ 3b) je složitost opět lineární.

Případ č. 4 nedělá žádné potíže (složitost je lineární), i přesto, že 71.1% buněk je prázdných.

Tabulka 2 obsahuje časovou závislost (v sekundách) jednotlivých data setů na parametrech k a n.

Tabulka 3 obsahuje časovou závislost (v sekundách) jednotlivých data setů na parametru k. Pro toto měření bylo použito $n = 10000$. Z této tabulky lze vidět, že pro malá k (1-5% z n) má algoritmus složitost nižší než lineární. Pro středně velká k (10-20% z n) má algoritmus lineární složitost a pro velká k (přes 20% z n) má algoritmus kvadratickou složitost.

Tabulka 4 ukazuje časovou závislost (v sekundách) jednotlivých data setů na měřítku mřížky(α). Z tabulky je zřejmé, že nejvýhodnější je volba $\alpha = 1.5$.

n	k	Data set 1	Data set 2	Data set 3a	Data set 3b	Data set 4
10000	10	0.1186	0.1286	0.1932	0.1108	0.1240
10000	50	0.1220	0.1296	0.1944	0.1176	0.1252
10000	100	0.1252	0.1328	0.1966	0.1120	0.1264
20000	10	0.2350	0.2494	0.4262	0.2196	0.2448
20000	50	0.2352	0.2536	0.4284	0.2218	0.2462
20000	100	0.2462	0.2560	0.4296	0.2242	0.2492
40000	10	0.4712	0.5074	0.9765	0.4404	0.4942
40000	50	0.4734	0.5130	0.9778	0.4426	0.4954
40000	100	0.4802	0.5152	0.9813	0.4460	0.4998
80000	10	0.9501	1.0326	2.5466	0.8942	1.0118
80000	50	0.9534	1.0345	2.5576	0.8954	1.0138
80000	100	0.9558	1.0404	2.5686	0.8964	1.0140

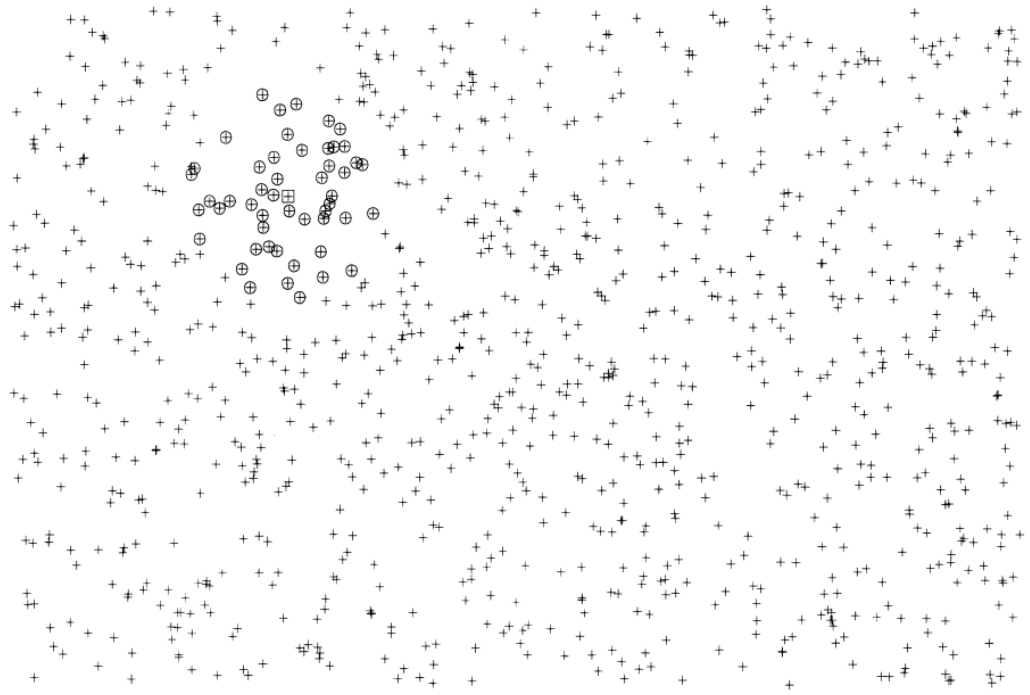
Tabulka 2: Časová závislost na parametrech k a n [4]

k	Data set 1	Data set 2	Data set 3a	Data set 3b	Data set 4
100	0.1164	0.1196	0.1802	0.1164	0.1264
200	0.1318	0.1276	0.1878	0.1254	0.1340
400	0.1494	0.1497	0.2132	0.1460	0.1494
800	0.2010	0.2000	0.2912	0.1988	0.2022
1600	0.4118	0.4030	0.5790	0.4008	0.3988
3200	1.3250	1.2896	1.6686	1.2950	1.2666
6400	5.0460	4.8026	5.2992	4.7840	5.0344

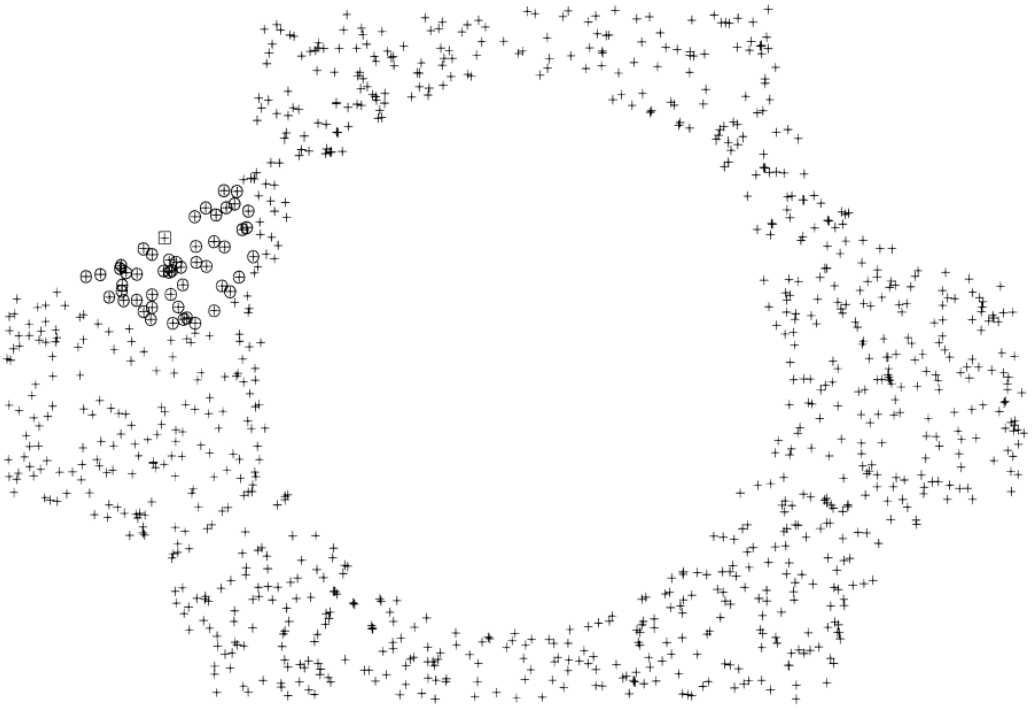
Tabulka 3: Časová závislost na parametru k [4]

α	Data set 1	Data set 2	Data set 3a	Data set 3b	Data set 4
0.25	1.0760	0.9314	4.7610	0.9568	0.8976
0.50	0.3290	0.2636	0.6590	0.2668	0.2614
0.75	0.1980	0.1548	0.2800	0.1528	0.1560
1.00	0.1176	0.1198	0.1812	0.1164	0.1274
1.25	0.1010	0.1108	0.1462	0.1032	**0.1208**
1.50	0.0934	**0.1086**	0.1308	**0.1000**	0.1264
1.75	**0.0924**	0.1164	**0.1274**	0.1032	0.1464
2.00	0.0956	0.1274	0.1306	0.1078	0.1834
2.25	0.0980	0.1482	0.1372	0.1164	0.2218
2.50	0.1054	0.1758	0.1494	0.1286	0.2450
2.75	0.1122	0.2032	0.1614	0.1428	0.2704
3.00	0.1208	0.2252	0.1802	0.1636	0.2932
4.00	0.1812	0.3142	0.2592	0.2428	0.4294
5.00	0.2450	0.4350	0.3384	0.3240	0.6174
6.00	0.3208	0.5811	0.4426	0.4252	0.8480
7.00	0.4022	0.7558	0.5624	0.5482	1.1062

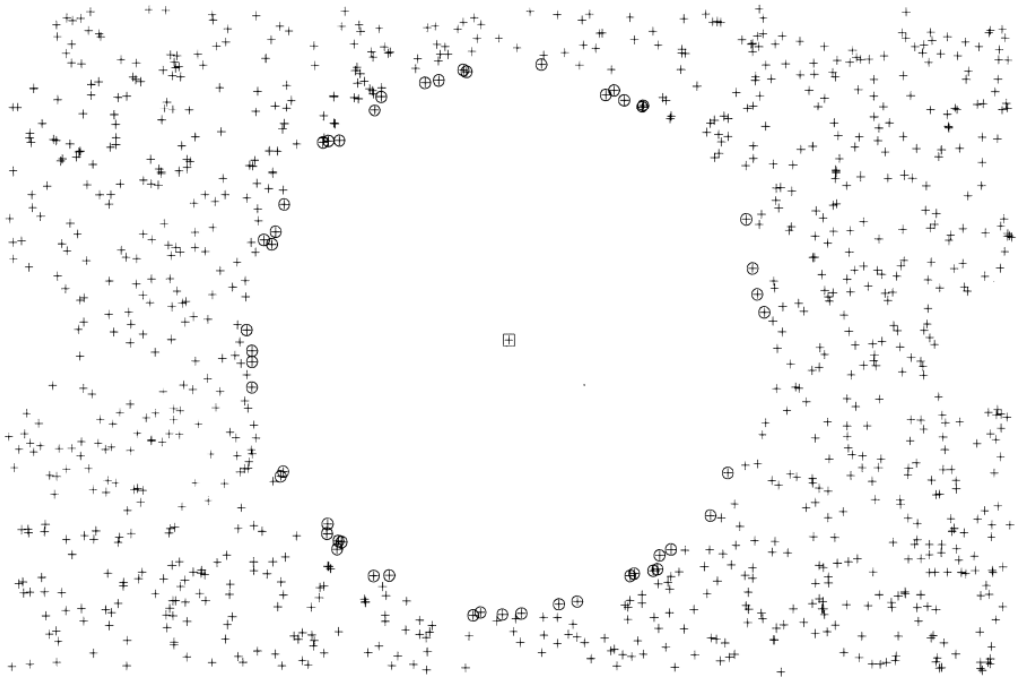
Tabulka 4: Časová závislost na měřítku mřížky [4]



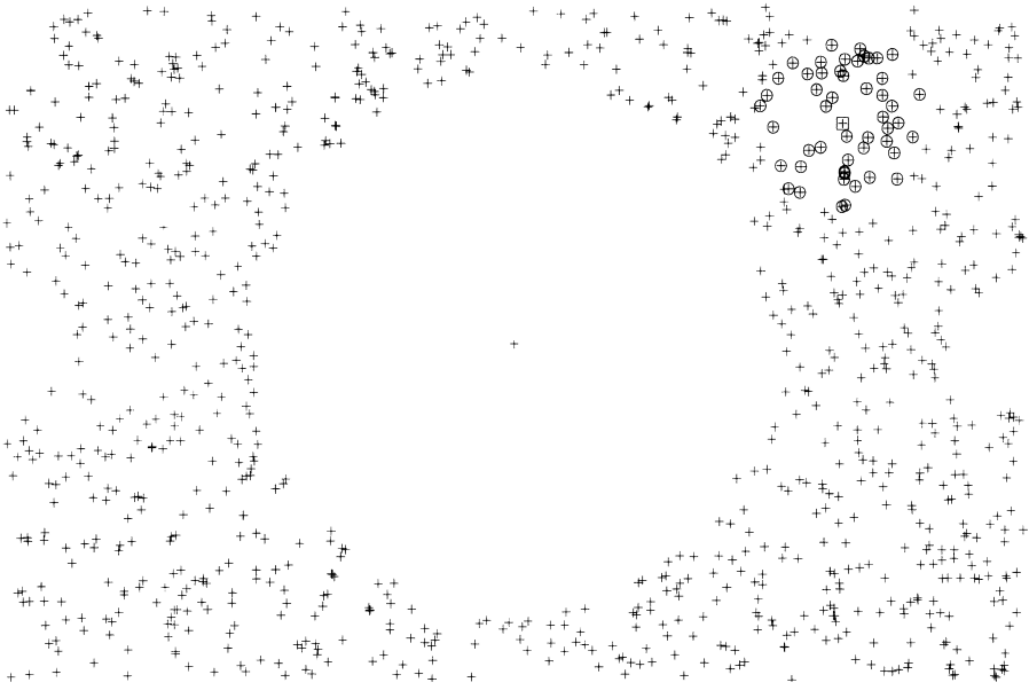
Data set 1: uniformně rozdělená data [4]



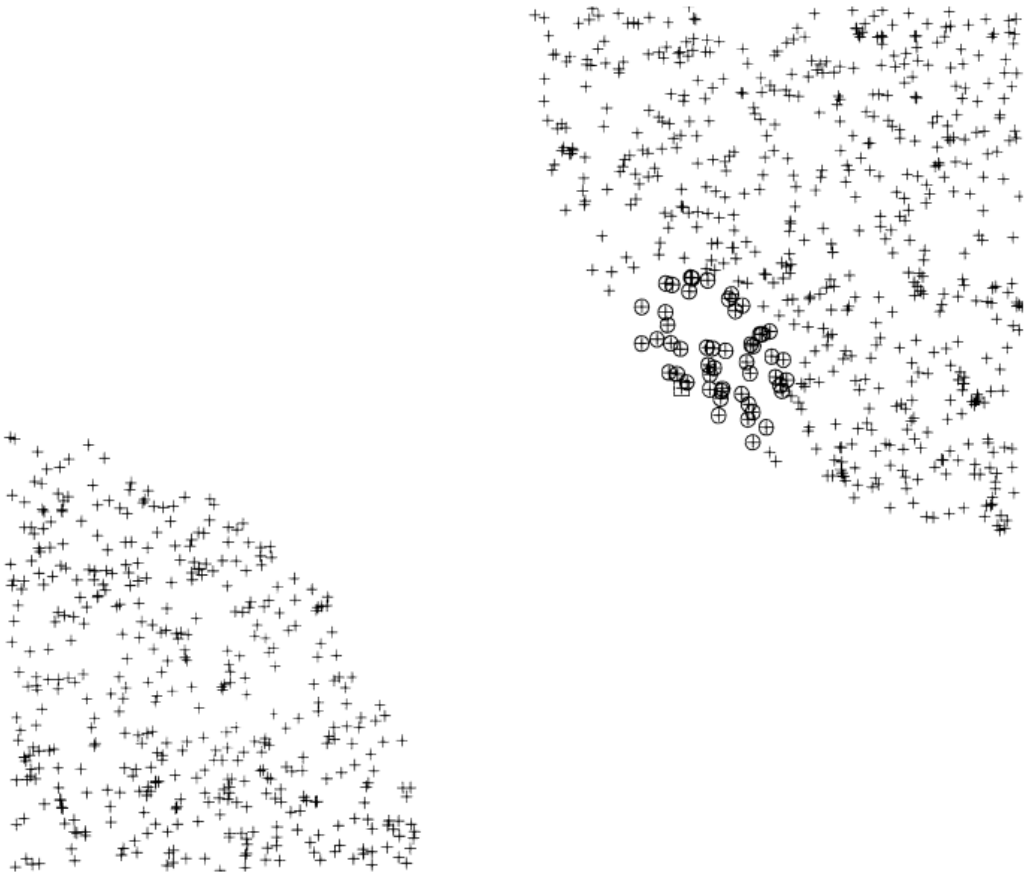
Data set 2: díra v datech [4]



Data set 3a: díra v datech, hledání probíhá z izolovaného bodu [4]



Data set 3b: díra v datech, hledání probíhá z bodu, který není izolovaný [4]



Data set 4: data jsou rozdělena do dvou množin a vyhledávání probíhá v jedné z nich [4]

Závěr

Představený algoritmus pracuje ve 2-D a je zřejmé, že je výhodné ho využívat pouze pro malé hodnoty k. Testy ukázali, že časová složitost se příliš nezmění v závislosti na topologii dat. Aby se výsledky testování daly reprodukovat, je důležité, aby programátor správně naimplementoval uniformní mřížku a správně pracoval s daty. Algoritmus nepotřebuje výrazně realokovat paměť. Lineární časová složitost lze očekávat až do 70% prázdných buněk v mřížce.

Literatura

[1] Edelsbrunner H. Algorithms in combinatorial geometry. New York: Springer-Verlag, 1987

[2] O'Rourke J. Computational geometry in C. New York: Cambridge University Press, 1994

[3] Preparata FP, Shamos MI. Computational geometry: an introduction. New York: Springer-Verlag, 1985 Wesley, Massachusetts, 2nd ed.

[4] Piegl, L.A. & Tiller, W. (2002). Algorithm for finding all k nearest neighbors. Computer-Aided Design 34 (2002) 167-172