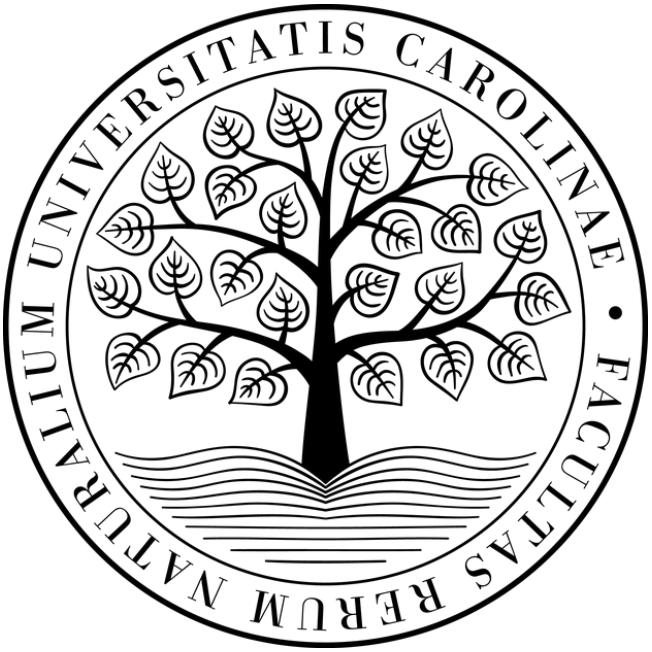
Univerzita Karlova

Přírodovědecká fakulta



Úvod do programování

Spatial sensitive hashing a approximate nearest neighbor search

Dokumentace programu

Filip Kradijan Seider

3. BGEKA

Jesenice

27. 2. 2023

1. Zadaní programu

Zadání je modifikací příkladu č. 58: Prostorové hashování (Spatial Sensitive Hashing) rozšířeného o vyhledávání odhadovaného nejbližšího souseda (*approximate nearest neighbor search* (ANNS)).

2. Rozbor problému

Hashování je výpočet krátkých hashovacích klíčů, tvořených sekvencí bitů, ze vstupních dat pomocí hashovacích funkcí. Jednodušší reprezentace původně obsáhlejších dat vede k úspoře paměti a rychlejšímu vyhledávání. Jedna z možností dělení hashovacích postupů je na prostorově citlivé hashování (*locality–sensitive hashing* (LSH)) a hashování založené na učení (*learning–based hashing* (LBH)).

Na rozdíl od LBH, LSH využívá hashovací funkce nezávislé na vstupních datech a generuje klíče / hashovací prostor takové/ý, aby pro blízké/podobné vstupy byla vysoká pravděpodobnost kolize. Tento program redukuje 2D souřadnice na prostorový hash (p. h.) reprezentující část 2D prostoru.

Vyhledávání nejbližšího souseda (*nearest neighbor search* (NNS)) je významným algoritmem v mnoha oblastech informatiky. Kvůli problémům s rychlostí vyhledávání nebo vysokou dimenzionalitou vstupních dat je NNS nahrazován ANNS. Principem ANNS je nalezení nejbližšího souseda s vysokou pravděpodobností místo s jistotou.

3. Existující algoritmy

[4] uvádí metody ANNS s využitím LSH modifikaného o následující koncepty: obecná odlišnost (*general dissimilarity*), znalost metriky (*metric learning*) a jádrové metody (*kernel methods*).

4. Popis zvoleného algoritmu

Algoritmus pracuje s n body. Okolo bodů v závislosti na rozdílech minimálních a maximálních souřadnic v daných dimenzích vytváří ohraničující pravoúhlý útvar (*bounding box*). Strany útvaru jsou rozděleny na k částí, v tomto případě nejbližšímu celému číslu ke $\sqrt[4]{n}$. Ohraničující čtyřúhelník tedy bude rozdělen na $k^{počet\ dimenzí}$ p. h., délky jejichž stran odpovídají vztahu:

$$dimenze_{interval} = |dimenze_{max} - dimenze_{min}| / k.$$

Následně je každý bod přiřazen do p. h. Každému bodu je určen ANN. Pokud se v p.h. bodu nachází další body, ANN je určen jako nejbližší bod z p. h., jinak je hledán v nejbližších ostatních p.h. obsahujících body.

Algoritmus vrací nejpřesnější výsledky pro data s pravidelným rozdělením, pokud všechny $dimenze_{interval}$ jsou stejné. Algoritmus z p. h. bodu vrací skutečného nejbližšího souseda, pokud pro vzdálenost bod–těžiště p. h. d platí:

$$d < \frac{vzdálenost\ těžiště - nejbližší\ hranice\ p.h.}{2}.$$

5. Struktura programu

Využité odvozené datové struktury: *seznam*, *n–tice*.

5.1. Třídy[1]

Bod – Point:

Datové položky třídy:

- count – počet objektů třídy *Point*

Datové položky objektu:

[1] přístup k veškerým proměnným i metodám všech tříd je nastaven na *private*, přístup k hodnotě všech atributů je možný díky využití dekorátoru @property místo funkce typu getter

- ID – identifikátor bodu
- x – x–souřadnice
- y – y–souřadnice
- hash – klíč p. h., do kterého bod spadá
- ANN – identifikátor odhadovaného nejbližšího souseda

Obdélník – Rectangle:

Datové položky objektu:

- x1– nejmenší x–souřadnice obdélníku
- x2 – největší x–souřadnice obdélníku
- y1 – nejmenší y–souřadnice obdélníku
- y2 – největší y–souřadnice obdélníku
- hash – klíč p.h., který objekt třídy *Rectangle* reprezentuje
- points – kolekce pro body spadající do daného p.h.
- addPoint(pnt) – do atributu *points* přidává bod
- center(self) – vrací bod reprezentující těžiště odbélníku
- pointCount(self) – vrací počet bodů spadajících do obdélníku

5.2. Funkce

Parametry programu – *parse*

Do vraceného objektu přiřazuje cesty k parametrům programu, pokud nebyly zadány, přiřazuje výchozí soubory.

Kontrola vstupních souborů – *control (pointInput, pointOutput, hashOutput)*

Pokud některý soubor, se kterým program pracuje, neexistuje, nejedná se o textový soubor nebo s ním nelze pracovat, s odpovídající zprávou ukončuje program.

Převod řádku na bod – *rowToPoint (line)*

Pokud je to možné, vytváří ze vstupního řetězce 2D bod.

Přidání bodu do seznamu – *appendPoint (row, pointList, xMinimum, xMaximum, yMinimum, yMaximum)*

Pokud pro vstupní řetězce jde vytvořit bod, přidá jej do seznamu bodů. Pokud souřadnice bodu jsou větší/menší, než zaznamenané extrémy, přepíše je.

Načtení bodů – *loadPoints (input file)*

S využitím podfunkcí *appendPoint* a *rowToPoint* po řádcích vstupního souboru načítá body, ty ukládá do seznamu. Zjišťuje extrémy souřadnic. Ukončuje program, pokud nešlo načíst žádný bod.

Vytvoření p. h. – *createHashes (points, xMinimum, xMaximum, yMinimum, yMaximum)*

Z extrémů souřadnic vytváří ohraničující pravidelný čtyřúhelník, rozděluje jej na p.h., ty jsou indexovány od 1 po řádcích zleva a zespoda.

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

Obrázek 1: indexování p. h. pro $k = 4$, zdroj: vlastní zpracování

Každou hranici v obou dimenzích pro zjednodušení další práce přidává do seznamu.

Rozdělení bodů do p. h. – *hashPoints (sideDivisions, points, bins, borders)*

Každému bodu v seznamu bodů zapisuje p. h., do kterého vkládá daný bod. P. h. musí zahrnovat celý ohraničující čtyřúhelník, včetně svých hranic., proto jsou intervaly p. h. určeny následovně:

Pro x–souřadnici:

$$\langle x_{min} + (sloucový\ index - 1) \cdot x_{interval};\ x_{min} + sloucový\ index \cdot x_{interval})$$

pro sloupcový index[2] jiný než *k*,

$$\langle x_{min} + (sloucový\ index - 1) \cdot x_{interval};\ x_{max}\rangle$$

pro sloupcový index roven *k*.

Pro y–souřadnici:

$$\langle y_{min} + (řádkový\ index - 1) \cdot y_{interval};\ y_{min} + řádkový\ index \cdot y_{interval})$$

pro řádkový index jiný než *k*,

$$\langle y_{min} + (řádkový\ index - 1) \cdot y_{interval};\ y_{max}\rangle$$

pro řádkový index roven *k*.

Vzdálenost dvou bodů – *pointDistance (p1, p2)*

Vzdálenost dvou bodů počítaná Pythagorovou větou.

Odhad nejbližšího souseda – *approximateNearestNeighbor (points, bins)*

Pokud seznam bodů obsahuje 2 a více bodů, přiřazuje každému identifikátor ANN, pokud neexistují další body, bodu ponechává výchozí hodnotu atributu ANN *-1*. Pokud se v p.h. bodu nachází další body, je mu přiřazen nejbližší bod v něm., jinak je z ostatních p.h., obsahujících body, vybrán ten s nejbližším těžištěm od těžiště p. h. bodu., případně všechny p.h. se stejnou vzdáleností. Z vybraných p.h. je bodu přiřazen

[2] sloupcový i řádkový index jsou indexovány jako p.h., tedy od levého dolního rohu ohraničujícího čtyřúhelníku

z nich nejbližší bod. Algoritmus zanedbává případ, kdy nejnižší vzdálenost k bodu má více bodů, jako ANN určuje první s touto vzdáleností.

Výpisy – *output (pntOutput, points, binsOutput, bins)*

Do výstupních souborů zapisuje vybrané atributy všech bodů ze seznamu bodů a všech p. h. z jejich seznamu tak, aby 1. znak atributů, jenž jsou všechny odděleny tabulátorem, vždy začínal stejnou pozicí v řádku.

Pracovní postup – *(pntInput, pntOutput, hashOutput)*

S využitím všech funkcí, kromě *parse*, vykonává celý proces.

6. Vstupní data[3]

Vstupní body

Program převádí každý řádek na bod, aby byl převod úspěšný, musí první 2 výrazy na řádku být čísla oddělená libovolným počtem bílých znaků (whitespace). Pokud je číslo desetinné, desetinnou část od celočíselné lze oddělit tečkou nebo desetinnou čárkou. Program vyžaduje právo číst soubor a prostorově unikátní body, celkem minimálně 2, pro správné fungování ANNS. Pro vytvoření více než 1 p.h. je potřeba alespoň 6 bodů.

Výstupní soubory

Program vyžaduje právo zapisovat do obou souborů. Pokud soubory neexistují, budou vytvořeny, pokud existují, jejich případný obsah bude nahrazen výstupními daty.

[3] veškeré vstupní soubory musí mít formát *TXT*

7. Výstupní data

Zpracované body

Každý úspěšně načtený bod je vypsán na samostatný řádek ve formátu:

ID x y hash ANN

P. h.

Každý p.h. je vypsán na samostatný řádek ve formátu:

ID x1 x2 y1 y2,

hranice p. h. reprezentují hranice vzniklé dělením ohraničujícího pravidelného čtyřúhelníku, viz funkce *createHashes.*

8. Spuštění programu

Program pro fungování vyžaduje umístění souboru *hashingObjects.py* do složky, do níž má přístup. Kromě interpreteru jazyka *Python jej lze* spustit přes příkazový řádek, v tom lze v libovolném pořadí zadat parametry programu:

- -p <cesta_K_SouboruVstupníchBodů> / *body.txt*[4]
- -op <cesta_K_SouboruZpracovanýchBodů> / *hashovaneBody.txt*
- -oh <cesta_K_SouboruVýstupníchHashů> / *hraniceHashu.txt*

Pro spuštění přes příkazový řádek je potřeba nastavit složku souboru přes příkaz *cd <cesta do složky programu>*. Poté lze program spustit pomocí příkazu *python main.py*, pro parametry *-p input.txt*, *-op output1.txt* a *-oh output2.txt* např. příkazem „*python main.py -p input.txt -op output1.txt -oh output2.txt*“.

[4] název výchozího souboru, který program hledá ve své složce, pokud nebyl zadán daný parametr

9. Porovnání s NNS

Pomocí funkce *random.uniform(0, 1000000)* byly vytvořeny vstupy s na sobě nezávislými souřadnicemi. Poté byl tento algoritmus porovnán s NNS, viz Tabulka 1.

počet bodů	rozsah. x–souřadnic	rozsah. y–souřadnic	zpracování pomocí p.h. a ANNS	zpracování pomocí NNS	odlišně určených NN [%]
10 tis.	*0–899 890,531822576*	0–999 979,329261538	1,54 s	78,42 s	5,86
100 tis.	0–944 441,3359049368	*0–999 999,2252278785*	26,68 s	120 min.	3,43

Tabulka 1: porovnání efektivity p.h. a ANNS s NNS, zdroj: vlastní zpracování

10. Možná vylepšení

Program by mohl pro maximální přesnost přesnost souřadnicových výpočtů používat knihovnu *Decimal*. Šlo by jej rozšířit o např.: práci s vícedimenzionálními daty, určení ANN s přesností zadanou uživatelem a systematické prohledávání vlastního p.h., pokud je bod blíže jeho těžišti, než jeho nejbližší hranici.

Dále by program mohl být flexibilnější, např. pracováním s dalšími formáty souborů (CSV, SHP, GeoJSON apod.).

11. Zdroje

[1] SLOAN, S., W. (1993): A fast algorithm for generating contrained Delaunay triangulations. Computers & Structures, 47, 3, 441–450.

[2] Stack Overflow (2010): Sort a list by multiple attributes?., https://stackoverflow.com/questions/4233476/sort-a-list-by-multiple-attributes (13.1.2023).

[3] LIEFU, A., JUNQING. Y., ZEBIN, W., YUNFENG, H., TAO, G. (2017): Optimized residual vector quantization for efficient approximate nearest neighbor search. Multimedia Systems, 23, 169–181.

[4] DING. K., HUO, C., FAN, B., XIANG, S. M., PAN, CH. (2018): In Defense of Locality-Sensitive Hashing. IEEE Transactions on Neural Networks and Learning Systems, 29, 87–103.

[5] PRICE, E. (2019): Lecture 28: Locality Sensitive Hashing, https://www.cs.utexas.edu/~ecprice/courses/randomized/fa19/notes/lec28.pdf (15.1.2023).