BI-VWM: R-strom

Adam Peňáz
Dávid Jenčo

Popis projektu

V tomto projektu jsme se zaměřili na výzkum a perzistentní implementaci indexové struktury R-strom. R-strom je datová stromová struktura využitelná pro vyhledávání n-dimenzionálních objektů. R-strom je často využíván v geografických informačních systémech a je tedy dobře aplikovatelný např. pro vyhledávání na webu v mapových systémech.[1]

Vstupem programu je dimenze stromu, ve které se vygenerují náhodná data a následně jednotlivé příkazy, které umožňují vkládat nové záznamy nebo vyhledávat ve stromu pomocí rozsahového nebo knn (k nearest neighbours) dotazu. Výstupem programu je v případě dotazu seznam ID položek, které danému dotazu odpovídají a potvrzení o zdaření operace v případě generování dat nebo vložení nového záznamu.

Za cíle projektu si klademe zejména:

- vlastní implementace této datové struktury (v její perzistentní podobě a s různými technikami dělení uzlů)
- porovnání rychlosti vyhledávání mezi R-stromem a sekvenčním hledáním (ať už rozsahovým či knn dotazem)
- porovnání dopadu techniky dělení uzlů na operace se strukturou
- porovnání rychlosti vyhledávání v závislosti na počtu dimenzí
- porovnání rychlosti vyhledávání v závislosti na počtu záznamů v jednom uzlu

Způsob řešení

R-strom jsme implementovali podle [2]. Ve stromu rozlišujeme dva typy uzlů - vnitřní a listové. Vnitřní uzly obsahují směrovací záznamy (routing entries), které popisují N-dimenzionální regiony a listové uzly pak obsahují jednotlivé koncové záznamy v tomto prostoru (ground entries), které si uchovávají identifikátory na vlastní data. V našem případě reprezentují tyto ID čísla řádků v datovém souboru, kde jsou data reálně uložena. Počet záznamů, který se vejde do listového uzlu, je větší než počet záznamů, který se vejde do vnitřního uzlu - jelikož v naší reprezentaci jsou koncové záznamy reprezentovány pomocí bodů (nikoli boxů), stačí, když mají uložených N (oproti 2N) rozměrů.

Techniku hledání vhodného listu při vkládání nových záznamů jsme zvolili single-way s následující heuristikou. Pro všechny směrovací záznamy v uzlu (routing entries) vypočítáme jejich vzdálenost od vkládaného záznamu. Z těchto vzdáleností vybereme tu nejkratší a pokračujeme v podstromu. V případě, že několik směrovacích záznamů má totožnou nejkratší vzdálenost, vybereme tu s nejmenším vnitřním obsahem (N-dimenzionální oblastí).

Maximální počet směrovacích/koncových záznamů (dále M) v jednom uzlu jsme museli obalit výpočtem, protože velikost směrovacího a koncového záznamu je jiná a abychom dosáhli stejné velikosti uzlu pro oba typy (vnitřní a listový), vypočítali jsme si nejmenší

[1] https://moodle-vyuka.cvut.cz/pluginfile.php/493984/mod_page/content/24/v-1.pdf
[2] https://www2.cs.sfu.ca/CourseCentral/454/jpei/slides/R-Tree.pdf

společný násobek směrovacího a koncového záznamu a z něj potom první násobek, který je větší než námi nastavená minimální velikost jednoho uzlu. Řekněme, že jsme si zvolili velikost uzlu alespoň 1024B, potom:

$$l = lcm(routingEntry, groundEntry)$$
$$k = ceil(1024/l)$$
$$nodeSize = k \times l$$

Přičemž minimální velikost jednoho uzlu jsme postupně navyšovali pomocí aritmetické řady v závislosti na použité dimenzi tak, aby se např. pro dimenzi 1 nestalo, že uzel bude mít stovky záznamů a naopak pro dimenzi 20 nebyly v uzlu jednotky záznamů. Námi optimálně zvolený minimální počet směrovacích záznamů je kolem 30 (počet koncových záznamů pak kolem 50).

Při přeplnění uzlu ($M + 1$ záznamů) používáme dvě politiky dělení. Náhodné přerozdělení záznamů a quadratic split [3]. Obě politiky mají ve svém závěru $ceil((M + 1)/2)$ záznamů v původním přeplněném uzlu a $floor((M + 1)/2)$ záznamů v novém uzlu.

Při serializaci jednotlivých uzlů vyplníme nezaplněný prostor uzlů umělým paddingem $(number\ of\ entries < M)$. Uzly potom sice zabírají o něco více místa než by museli, ale dosáhneme tím možnosti rychlé navigaci v binárním souboru.

$$position = metadataSize + id \times nodeSize$$

Implementace

Pro vývoj jsme na základě doporučení použili jazyk C++, se kterým jsme měli zatím nejvíce zkušeností, co se týče programů s více zdrojovými soubory a prací s binárními soubory. Taky jsme díky nízkoúrovnosti jazyka měli větší kontrolu nad odkazy v paměti a efektivitou algoritmů.

Z knihoven jsme používali hlavně standardní knihovnu stl a taky knihovnu filesystem. Při měření se nám potom hodila knihovna chrono nebo cassert pro automatické testy.

U automatizovaných testů jsme pro dimenze 1-10 zkusili vždy porovnat výsledky 1000 náhodných rozsahových dotazů pomocí R-stromu se stejným dotazem pomocí sekvenčního průchodu a navíc taky 1000 náhodných knn dotazů (opět R-strom vs sekvenční průchod).

[3] https://www.just.edu.jo/~qmyaseen/rtree1.pdf

```
bool testOneKnn(RTree & tree, DataGenerator & generator, int dimension){
    vector<int32_t> point;
    point.reserve(dimension);

    for (int i = 0; i < dimension; ++i) {
        point.emplace_back(generator.getRandomInt());
    }

    set<KnnSearchStruct> result1;
    set<KnnSearchStruct> result2;
    tree.knnSearch(point, k: 20, &: result1);
    doTheKnnSearch(point, k: 20, &: result2);

    double max = result1.rbegin()->distance;
    result1.erase(first: result1.lower_bound(x: KnnSearchStruct(id: 0, max)), result1.end());
    result2.erase(first: result2.lower_bound(x: KnnSearchStruct(id: 0, max)), result2.end());

    return result1 == result2;
}
```

Výsledky Knn dotazů u R-stromu a sekvenčního průchodu se nemohou porovnávat přímo, protože pokud poslední K-tý soused leží ve stejné hranici s jiným záznamem (range query dotazu) nemáme jistotu, že R-strom i sekvenční průchod budou mít stejný výsledek, protože korektní budou všechny body ležící v této hranici. Proto před tím než výsledky porovnáme, všechny body ležící na hranici smažeme a porovnáme zbytek, který už je jednoznačně určený.

Příklad výstupu

Na následujícím screenshotu můžeme vidět příklad práce s programem. Nejdříve se vygenerovala náhodná data požadované dimenze (počet záznamů je hardcoded) a následně se provedl rozsahový dotaz, za kterým následovalo vložení nového záznamu.

```
generate
Enter dimension:
3
Done
search range
Enter ranges for dimension 3
-1000 2000 100 3510 -4000 90
2608 9750 15719
insert
Enter point ranges for dimension 3
2000 4010 -150
Done
quit
```

Experimentální sekce

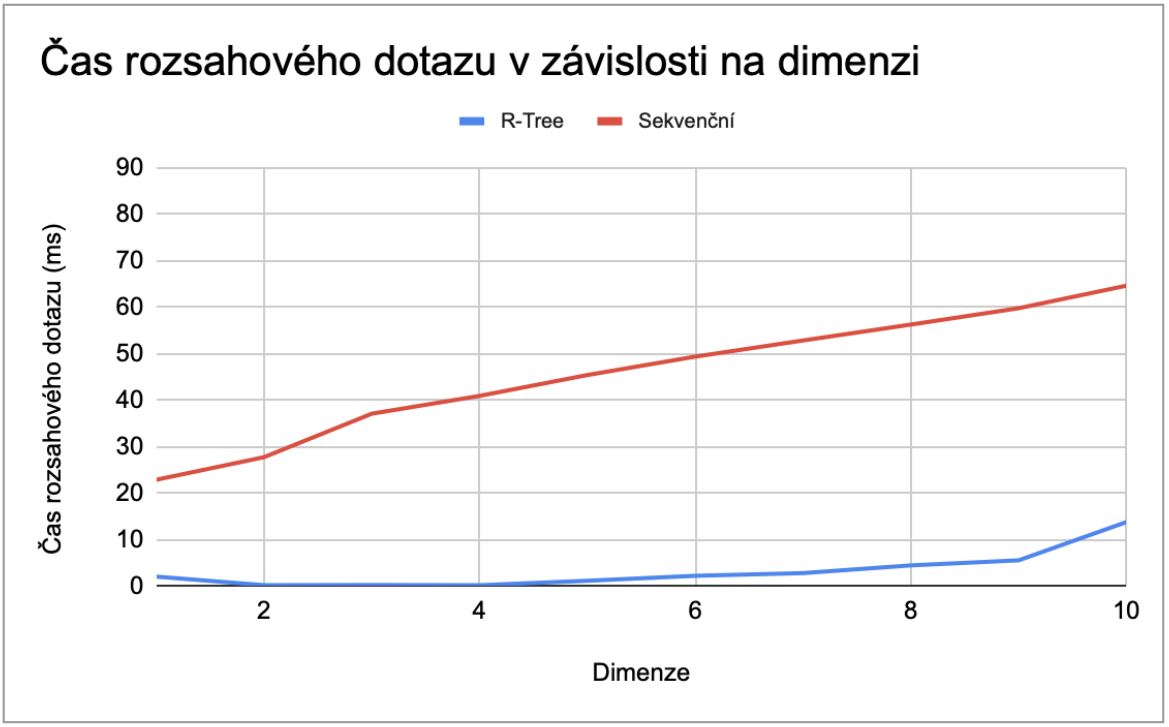
Všechna měření jsme prováděli na procesoru Intel Core i7-8565U, 1.8GHz, 4 jádra. U všech experimentů používáme při vkládání záznamů (a tedy generování dat) kvadratický algoritmus dělení, pokud není uvedeno jinak.

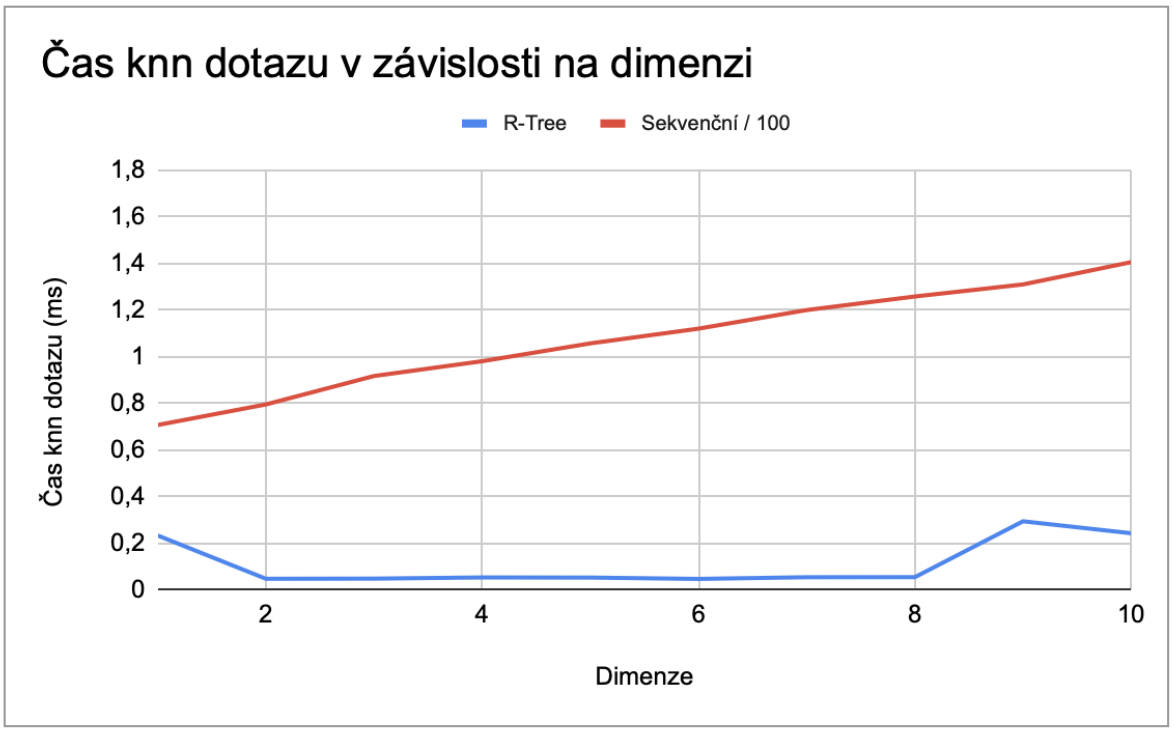
Ukázka měření doby rozsahového dotazu v závislosti na dimenzi:

```
for (int dimension = 1; dimension <= 10; ++dimension) {
    DataGenerator generator (dimension, DATA_PATH);

    double searchRangeTime = 0;
    generate( &: tree,  &: generator, dimension);

    for (int i = 0; i < numberOfQueries; ++i) {
        searchFrom.clear();
        searchTo.clear();
        for (int j = 0; j < dimension; ++j) {
            searchFrom.emplace_back(getRandomInt( from: -5000,  to: -2000));
            searchTo.emplace_back(getRandomInt( from: 2000,  to: 5000));
        }
        clock.tick();
        tree.rangeSearch(searchFrom, searchTo);
        clock.tock();
        searchRangeTime += (double)clock.duration().count() * 1e-6;

    }
    searchRangeTime /= numberOfQueries;
    cout << dimension << ": " << searchRangeTime << endl;
}
```

Porovnání času vyhledávání pomocí R-stromu a sekvenčního průchodu

V tomto měření jsme porovnávali rychlost hledání v závislosti na dimenzi a v jednom grafu jsme vykreslili průběhy pro hledání za pomocí R-stromu a pomocí sekvenčního průchodu. Bylo generováno 20 000 položek dat v rozmezí -5 000 až 5 000. V každém vygenerovaném datasetu v dané dimenzi jsme následně měřili čas pro 250 náhodných dotazů, které ale byly cíleny na menší plochu, aby jeden dotaz vracel v průměru 10 výsledků. Z celkového času všech dotazů se následně vypočítal aritmetický průměr podle známého vzorce. Během vyhledávání v R-stromu byla použita cache pro uchovávání záznamů o velikosti 0.3MB.





U obou typů dotazů jde vidět zrychlení hledání pomocí R-stromu. I když rozdíl možná není tak markantní, jak bychom mohli očekávat, předpokládáme, že s daty o velikosti řádově GB by rozdíl byl daleko znatelnější. Taky jsme v měření zaznamenali náročnější průběh během prvního měření, což si vysvětlujeme tím, že cache stromu ještě nebyla optimálně naplněná. V měření knn dotazu jsme čas pro sekvenční průchod vydělili 100, abychom mohli vidět trendy obou křivek.

Závislost rychlosti vyhledávání na hustotě dat

V tomto experimentu jsme zkusili, jaký bude mít důsledek rozprostření dat po prostoru (hodnoty jednotlivých bodů) na rychlost vyhledávání (přičemž rychlost vyhledávání je zde závislá na počtu vložených dat). Zkusili jsme jak se zvyšuje čas potřebný pro vyhledávání s rostoucím počtem záznamů pro řídká data (hodnoty jednotlivých dimenzí byly od -300 000 do 300 000) a následně pro hustší data (hodnoty jednotlivých dimenzí byly od -5 000 do 5 000). Tyto dva časové trendy jsme následně vložili do poměru. (Bylo měřeno vždy 1000 rozsahových dotazů pro dimenzi 3.)

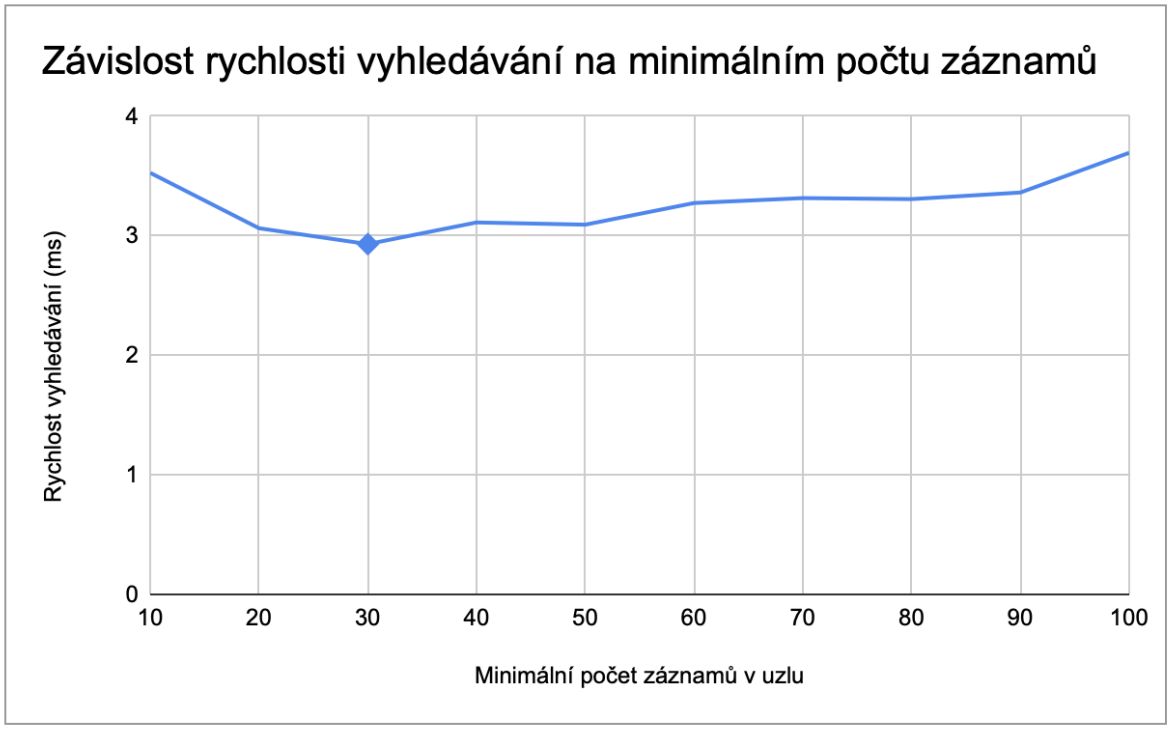
Můžeme vidět, že pro hustší data bylo hledání pro postupně se zvyšující množství dat mnohem rychlejší (peak pro 80 000 dat). To si vysvětlujeme tak, že pro řidší data dojde časem k pokrytí mnohem větší tzv. “mrtvé plochy” a tím ke snížení efektivity hledání v R-stromu. Naopak pro hustší data jsou bounding boxy více naplněné a nedochází k pokrytí těchto slepých míst.

Závislost počtu prohledaných uzlů na technice dělení uzlu

Zde jsme porovnali efektivitu kvadratického dělení při přeplnění uzlu oproti náhodnému dělení. Měřili jsme počet prohledaných uzlů v závislosti na zvyšujícím se množství vložených dat.

Tady je výsledek zřejmý - počet prohledaných uzlů se náhodným dělením zvyšuje a vyhledávání je tak méně efektivní. Navíc se zvyšuje pokrytí tzv. “mrtvé plochy” a překryv jednotlivých záznamů.

Závislost rychlosti vyhledávání na minimálním počtu záznamů v uzlu



Diskuze

V C++ jsmě měli největší potíže s počátečním zprovozněním file streamů a jejich správného fungování. Nejdříve jsme používali dva file streamy (input a output), které přistupovaly do stejného souboru. To se neukázalo jako šťastné řešení a nějakou dobu nám trvalo, než jsme naimplementovali korektní fungování obecnějšího fstream.

V naší implementaci jsme úplně a korektně nevyřešili obalující výpočet kolem vypočtení velikosti jednoho uzlu vzhledem k minimálnímu počtu záznamů v jednom uzlu, protože při vyšších dimenzích (cca 40+) je nejmenší společný násobek routing a ground entry příliš velký na to, aby dělení vyšlo nějak rozumně. Toto by se dalo vyřešit např. otočením přístupu tak, že bychom nejdřív zvolili optimální počet záznamů pro vnitřní a listový uzel a následně z tohoto údaje dopočítali velikost jednoho uzlu.

Multiway v naší implementaci rovněž není vypracovaný, protože na něj nezbyl čas, jelikož jsme se spíše zaměřili na různou politiku dělení uzlů.

Závěr

Naimplementovali jsme indexovou strukturu R-strom v její perzistentní podobě. Dále jsme implementovali možnosti vkládání nových prvků, rozsahového a knn dotazu. V měření jsme porovnávali zejména rychlosti vyhledávání (R-strom vs sekvenční průchod, řídká vs hustá data…), metodu dělení uzlů a nebo minimální počet záznamů v jednom uzlu.

Reference

R-Tree, https://www2.cs.sfu.ca/CourseCentral/454/jpei/slides/R-Tree.pdf. Accessed 1 May 2022.

Guttman, Antonin. “R-tree.” *Wikipedia*, https://en.wikipedia.org/wiki/R-tree. Accessed 1 May 2022.

“Nearest Neighbor Queries Outline.” *InfoLab*, 3 September 2009, https://infolab.usc.edu/csci599/Fall2009/slides/Nearest%20Neighbor%20Queries%20Slides.pdf. Accessed 1 May 2022.

“A new enhancement to the R-tree node splitting.” *Jordan University of Science and Technology*, https://www.just.edu.jo/~qmyaseen/rtree1.pdf. Accessed 1 May 2022.

Novák, Jiří. “An Application of Metric Indexing Methods to the Mass Spectrometry Data.” https://moodle-vyuka.cvut.cz/pluginfile.php/493984/mod_page/content/24/novakj4_2008dipl.pdf.