R-tree

Pavel Jahoda

April 25, 2018

Popis Projektu

Cílem projektu je vytvoření vlastní perzistentní implementace R-stromu, což je stromová struktura pro vyhledávání n-dimenzionálních objektů často využívaná v geografických informačních systémech. Řešení překonává požadavky zadání a kromě dotazů nad databází 2D nebo 3D objektů, zvládá i požadavky na n-dimenzionální objekty. Dotazovat se lze jak na nejbližšího souseda, tak na rozsahových dotazem (tj. které objekty jsou K jednotek daleko od bodu X). Aplikace obsahuje testovací modul, který si dokáže generovat náhodná data tak i pracovat s ručně zadanými daty. Testovací modul kontroluje vyváženost stromu, korektnost velikosti bounding boxů, také testuje dotazování na nejbližšího sousedu a rozsahové dotazy

Navíc oproti zadání, které umožnuje generování náhodných dat (které generuji v testovacím modulu), stahuji data z aplikace flightradar od která s 5 minutovým spožděním získávám data o letištích a také letadlech, která jsou v daný moment ve vzduchu. Kromě polohy získávám rychlost, typ letadla, leteckou společnost ke které letadlo patří a další zajímavé údaje. Kromě náhodných dat je tedy možné se dotazovat i na letecká data.

Způsob řešení

V této sekci se především zaměřím na popis vkládání prvku z čehož je pochopitelný celý princip algoritmu. Když chceme vložit prvek do stromu nacházíme se nejprve v kořenovu stromu (root). Pokud má kořen jako potomky listy, stane se náš prvek potomkem kořene. Pokud nemá kořen listy zjistíme kolika potomkům by se při vkládání našeho prvku nemusel zvětšovat minimal bounding box (prvek je uvnitř bounding boxu). Pokud je takových potomků více než jeden vnoříme se do toho, který má střed minimal bounding boxu nejblíž našemu vkládánmu prvku. V případě že je takový potomek jeden vnoříme se do něj a pokud takový potomek neexistuje, vnoříme se do potomka, kterému by se objem minimal bounding boxu zvětšil po přidání prvku nejméně. Takhle se budeme vnořovat dokud nedojdeme do uzlu který obsahuje listy a do něj prvek vložíme

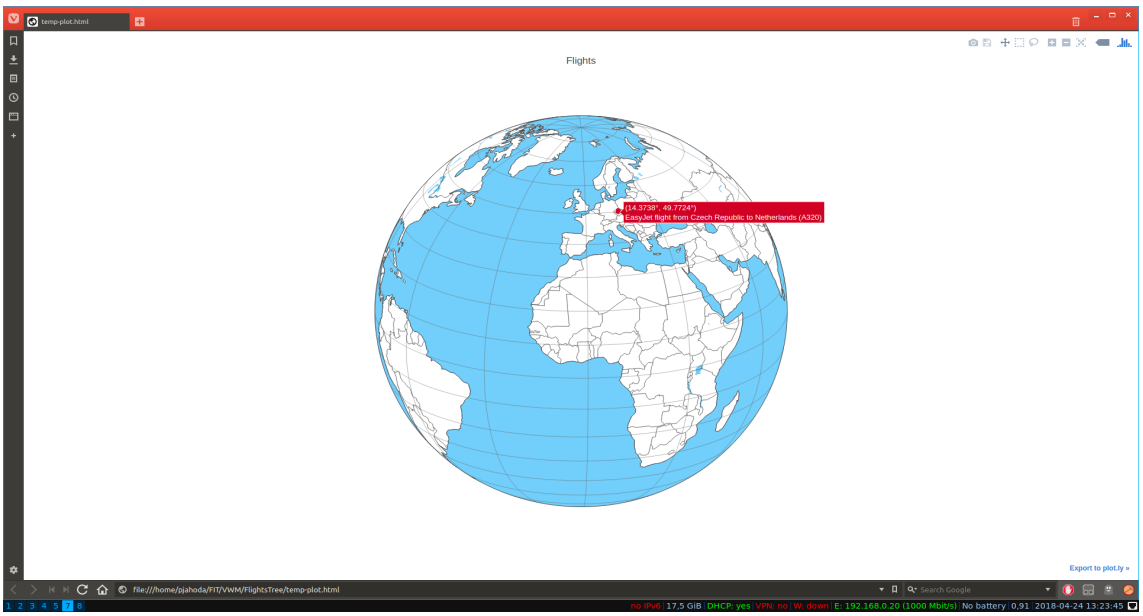
Při vkládání prvku do uzlu se může stát, že uzel obsahuje maximum námi nastaveného počtu potomků, v takové případě dochází na dělení uzlů. Z jednoho uzlu s 4 potomky a 1 vkládáným prvkem se může teoreticky stát 2 nové uzly, kde jeden má 2 potomky a druhý tři potomky. Jeden z nových uzlů se posunul na místo předchozího starého uzlu a druhý uzel se stává ”prvkem” vkládáným do předka našeho starého uzlu. Takovým způsobem se můžeme dostat až do kořenového uzlu, kde kořen rozdělíme na dva nové uzly jejichž předkem se stane úplně nový kořen.

V aplikaci je možné použit tři různé algoritmy na dělení uzlu. První algoritmus je náhodný, který zajistí pouze to, že nové uzly budou mít splněné podmínky na minimální a maximální počet potomků. Druhým algoritmem je řešení hrubou silou. Toto řešení prochází všechny validní permutace rozdělení uzlu a vybere takové, které vytvoří uzly s nejmenším součtem objemů minimal bounding boxů. Třetí řešení je variace[1] na quadratic split. Stejně jako v quadratic split algoritmu se nejprve vyberou dva prvké které není mít výhodné ve stejném uzlu a ty budou prvními prvky nových uzlů. První z vybraných dvou prvků bude mít nejmenší součet souřadnic a druhý prvek naopak největší součet souřadnic, takový to výběr prvků není optimální, nicméně je jeho asymptotická složitost pouze O(N) narozdíl od algoritmu Quadratic split, který má časovou asymptotickou složitost výběru prvních dvou prvků $O(N^2)$[?]. Zbytek prvků se přerozdělí tak, že se vybere m-1 (kde m je minimální počet potomků uzlu) prvků které jsou nejblíže k prvnímu vybranému prvku a přidají se do uzlu společně s ním. Stejně se vybere m-1 prvků k druhému prvku a zbytek se rozdělí podle toho, jestli je daný prvek blíž k prvnímu nebo druhému prvku.

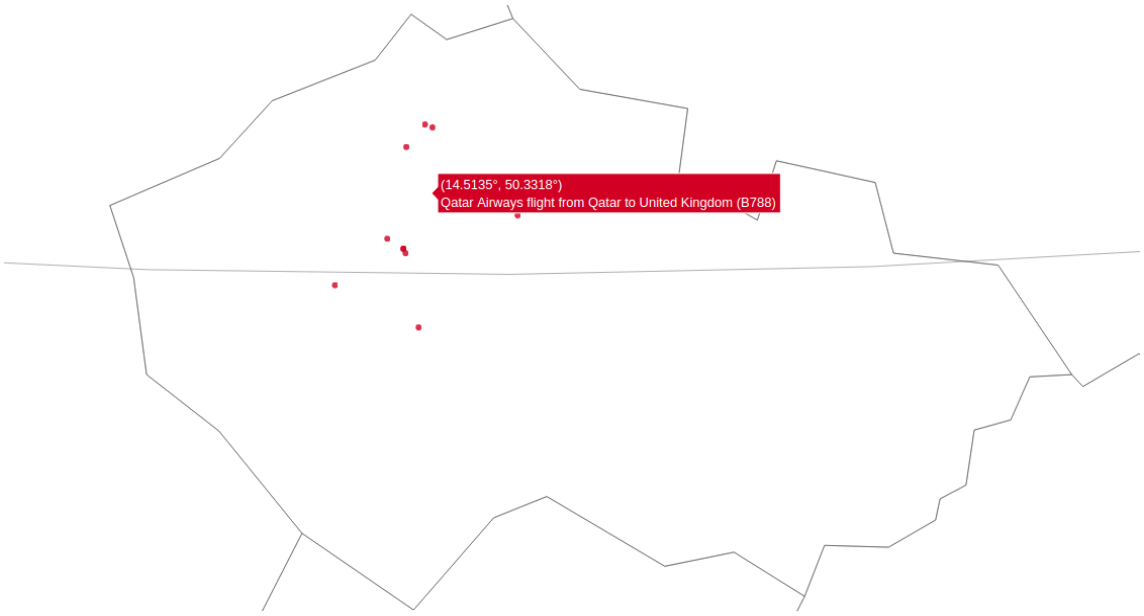
Implementace

Projekt byl programovaný v jazyce Python3 a použita byla knihovna numpy. Na grafické výstupy byla použita knihovna plotly (pip3 install plotly) a flightradar24 (pip3 install flightradar24) (na letecká data). Na znázornění grafů v experimentální sekci byla použita knihovna matplotlib. Pro grafické výstupy se spouští třída main (python3 main.py). Samotný R-strom lze testovat pouze s nainstalovaným jazykem Python3 voláním třídy RTreeTestCases (python3 RTreeTestCases.py).

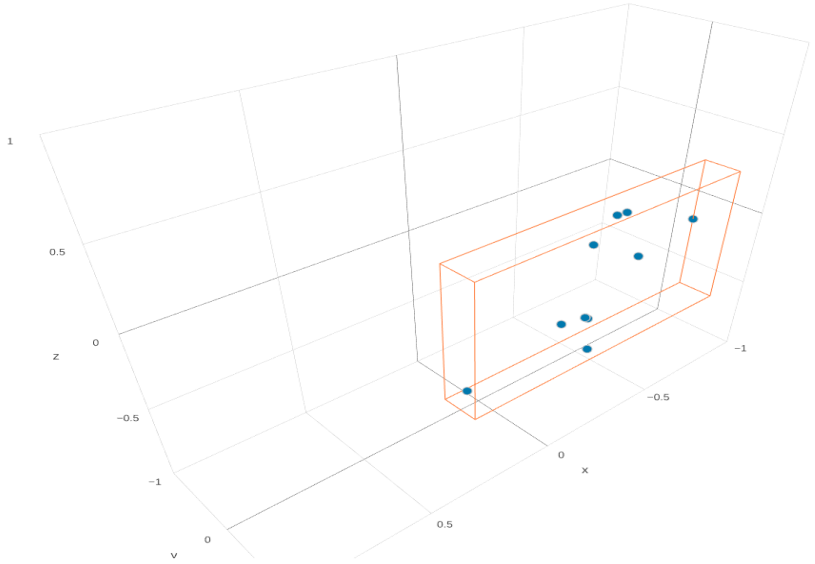
Příklad výstupu



Jak je patrné na obrázku výše jedná se o webové grafické rozhraní. Obrázek ukazuje výstup rozsahového dotazu na okolí Prahy.



Zde je přiblížený výstup k předchozímu dotazu. V obrázku je vidět, že aplikace ukazuje ke každému nalezenému letadlu odkud kam letí a k jaké letecké společnosti patří.



Výstup je také možné zobrazit ve 3D prostoru.

Experimentální Sekce

R-strom nabízí hned několik věcí na testování. Z hlediska rychlost vkládání prvků a následných dotazů můžeme porovnávat vliv různých metod dělení ůzlů, počet potomků uzlu, počet dimenzí nebo vzdálenost u rozsahového dotazu.

Tabulka níže (Table 1) ukazuje vliv volby metody dělení ůzlů (když uzel dosáhl maximálního množství potomků) na rychlost vkládání prvků a rychlost rozsahových dotazů. Data v tabulce jsou zprůměrováná z pěti různých testování na stromu s maximálním množstvím 8

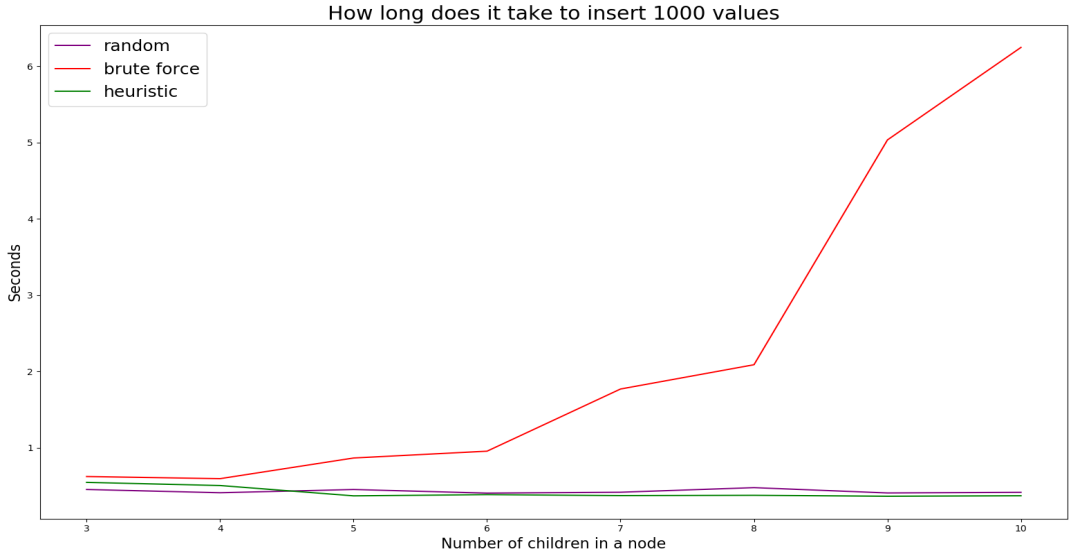
potomků na uzel, na trojdimenzionálních datech na 2 000 datech a 20 000 dotazech. Data jsou v sekundách.

Table 1: Rychlost vkládání a dotazů

	Insert	Query
Random	1.0669	19.45292
Brute force	4.73896	5.33882
Heuristic	0.91115	7.51662

Výsledek je očekávaný. Metoda hrubou silou (brute force) je jasně nejpomalější pří vkládání, zatímco na dotazech vyhrává. Metoda dělení heuristikou nejenže nejrychleji vkládá prvky, ale vede si obstojně i v rozsahových dotazech, což je důvodem proč jsou heuristiky na R-stromech používané. Na druhou stranu metoda náhodného rozdělení uzlů je suveréně nejhorší na rozsahové dotazy.

Na grafu níže můžeme pozorovat vliv počtu potomků uzlu na dobu vkládání prvků. Zatímco vkládání s dělením ůzlu heuristikou nebo náhodným dělením je s vzrůstajícím počtem potomků dokonce neptrně rychlejší tak u dělení hrubou silou můžeme pozorovat velký pokles rychlost přidávání. Tento fakt je způsoben tím, ze řešení hrubou silou skouší všechny možné permutace rozdělení uzlu, což je asymptoticky exponenciální.



Na závěr experimentální sekce bych zde uvedl, že počet dimenzí hraje naprosto minimální vliv na rychlost vkládání nebo na rychlost rozsahových dotazů. Toto zjistění potvrzuji silné stránky R-stromu.

Diskuze

Aplikace v momentální verzy neobsahuje plně funkční grafické uživatelské rozhraní(dále jen GUI). Momentálně se ve třídě main zavolá funkce, které se předá latitude a longitude a

vzdálenost od tohoto místa a aplikace najde veškeré letadla v této oblasti. Většina omezení, případně nedokonalostí projektu byla způsobena především tím, že jsem pracoval sám na projektu určeném pro dvě osoby. Tvorba plně funkční aplikace s GUI by byla nad rámec tohoto předmětu. V grafické verzy je latitude a longitude přepočítáváno na x,y,z validním způsobem pokud by země byla perfektní koule, což není. Naštěstí tento nepřesný výpočet nezpůsobuje příliš velké odchylky, takže přestože problém existuje - není na první pohled viditelný.

Závěr

Experimenty potvrdili výhody a nevýhody různých typů dělení uzlů R-stromu. Dále také potvrdili, že R-strom si zachovává efektivitu s rostoucím počtem dimenzí objektů. Do budoucna projekt nabízí možnost rozširitelnosti v podobě například lepšího GUI.

References

[1] A. F. Al-Badarneh, Q. Yaseen and I. Hmeidi, *A new enhancement to the R-tree node splitting*, (Journal of Information Science, 2009)

[2] A. Guttman *R-trees. A Dynamic Index Structure For Spatial Searching*, ACM 1984