# Zpráva k projektu - R-strom

Vincent Jakl (jaklvinc)
Matouš Kulhan (kulhama7)

## 1 Popis projektu

Cílem projektu bylo vytvořit persistentní implementaci R-stromu a k tomu uživatelskou aplikaci, která by umožnila nad R-stromem provádět dotazy a vkládat data. Naše řešení tyto požadavky splňuje a překonává. R-strom lze vytvořit jak v paměti, tak odkládaný na disk. Přidružená aplikace neumožňuje jen pracovat s jedním R-stromem, ale umožňuje i R-stromy vytvářet a přepínat mezi nimy. Nad konkrétním R-stromem lze vyhledávat, jak kNN, tak i rozsahovým dotazem a zároveň lze u 1D, 2D a 3D R-stromů zobrazit vizualizace.

## 2 Způsob řešení

Většina využitých algoritmů vychází přímo z originálního paperu Antonina Guttmana.

R-strom se skládá z dílčích uzlů. Každý uzel obsahuje jeden ze dvou typů záznamů: listy, nebo záznamy, které odkazují na další uzel.

- Listy vždy ukládají samotné záznamy. A to jako pole souřadnic a celočíselný klíč který je unikátní pro každý záznam.
- Záznamy, které odkazují na další uzly si udržují dvojici souřadnic, které tvoří obalující box pro všechny záznamy v uzlu, na který ukazuje. Dále si také udržují odkaz na svého potomka. V naší implementaci se jedná pouze o identifikační číslo, na kterém je potomek uložen. Tento odkaz se dá ale udržovat i například jako ukazatel na potomka.

Hlavní operace nad R-stromem jsou vkládání, mazání a vyhledávaní. V naší impementaci jsme se zaměřili pouze na vkládání a vyhledávání.
Při vkládání nového prvku do stromu vždy začínáme v kořeni. Pokud se jedná o uzel, který ukládá záznamy listů, tak záznam jen přidáme do uzle. Pokud se jedná o uzel, který ukládá odkazy na další uzle, tak si pro každý jeho záznam spočteme, o kolik by se obalující box daného uzle zvětšil. Jakmile nalezneme box s nejmenším zvětšením, tak se zanoříme do uzle, na který odkazuje. Pokud by se velikost zvětšení rovnala, vybereme ten s menší plochou. Tento krok opakujeme, dokud se nedostaneme na uzel, který ukládá listy. Do tohoto uzle zase přidáme náš záznam.

Dříve nebo později se dostaneme do situace, kde do uzle vložíme záznam a překročíme tím jeho maximální velikost. Tento uzel je poté třeba rozdělit. Rozdělujeme vždy záznamy do dvou uzlů, jeden již máme a druhý si vytvoříme. Do nového uzle přidáme alespoň jeden záznam, čímž zajistíme, že uzle již nebudou přetékat. Dělení uzlů je operace, pro níž existuje více řešení. V našem projektu jsme implementovali 3 různé typy dělícího algoritmu:

1. Dělení hrubou silou
   Dělení hrubou silou je nejprimitivnější dělení. V tomto dělení se porovnají všechny možné rozdělení a vybere se to, které vytvoří obalující boxy s nejmenší velikostí po sečtení. Tento algoritmus je nejpřesnější a zaručeně vybere vždy nejoptimálnější rozdělení. Je ale také nejpomalejší s rychostí $O(2^n)$, kde $n$ uvažujeme maximální počet prvků v uzlu.
2. Kvadratické dělení
   U kvadratického dělení si ze všech prvků vybereme 2, které tvoří největší obalující box. Tyto 2 záznamy přidám každý do jedoho nového uzlu. Poté si pro všechny ještě nepřidělené záznamy spočítáme o kolik by zvětšily obalující box prvního a druhého uzle. Ten u něhož bude rozdíl mezi zvětšením přidáme do uzle, který zvětší méně. Takto pokračujeme posupně se všemi zbylými záznamy. Tento algoritmus je obecně nejvíce používaný. Má dobrý poměr mezi optimalizací rozdělení a rychlostí. Má rychlost $O(n^2)$

3. Lineární dělení
   U lineárního dělení si vybereme pro každou dimenzi 2 vstupy a to tak, že pro první vstup platí, že v dané dimenzi jeho pravá hrana je v dimenzi nejvíce vlevo, a pro druhý platí, že jeho levá hrana je v dané dimenzi nejvíce vpravo. Z těchto dvojic si vybereme takovou, kde (vzdálenost pravé hrany prvního vstupu od levé hrany vstupu druhého / maximální vzdálenost mezi vstupy v dané dimenzi) je ta nejmenší. Ty zase rozdělíme do dvou nových uzlů. Zbytek vstupů postupně přiřadíme vždy do uzle, který zmenší nejvíce. Tento algoritmus je sice nejrychlejší, efektivita rozdělení není ale tak vysoká a rozdělené uzly se mohou jednoduše překrývat. Jeho rychlost je $O(n)$

Po vložení prvku je potřeba upravit všechny rodiče, který na něj postupně odkazují. Tento krok je poměrně jednoduchý a jen zvětší obalující box každého z rodičů při vracení z rekurze. Je třeba ale mít na vědomí, že pokud se uzel rozdělí, je nutné do uzle rodiče přidat nový prvek s obalujícím boxem nového uzlu a s odkazem na něj. Pokud by jeho přidáním uzel rodiče přetekl, je potřeba i na něj zavolat dělící algortimus. Takto se postupně postupuje zpět do kořene a R-strom se tím upraví opět do funkční podoby.

V našem projektu jsme také implementovali vyhledávání a to hned 2 typy. Vyhledávání všech prvků v daném rozmezí a vyhledávaní k nejbližších prvků k zadanému bodu. Při vyhledávání prvků v rozmezí začneme zase v kořeni stromu. Narozdíl od vkládání se ale zanoříme do všech uzlů, které se přesahují s naším search boxem. Tento krok zase opakujeme dokud se nedostaneme k uzlu s listy. Tam už vybereme všechny které patří do hledaného boxu a přidáme je do výsledku. O kNN vyhledávání je více v sekci Diskuze.

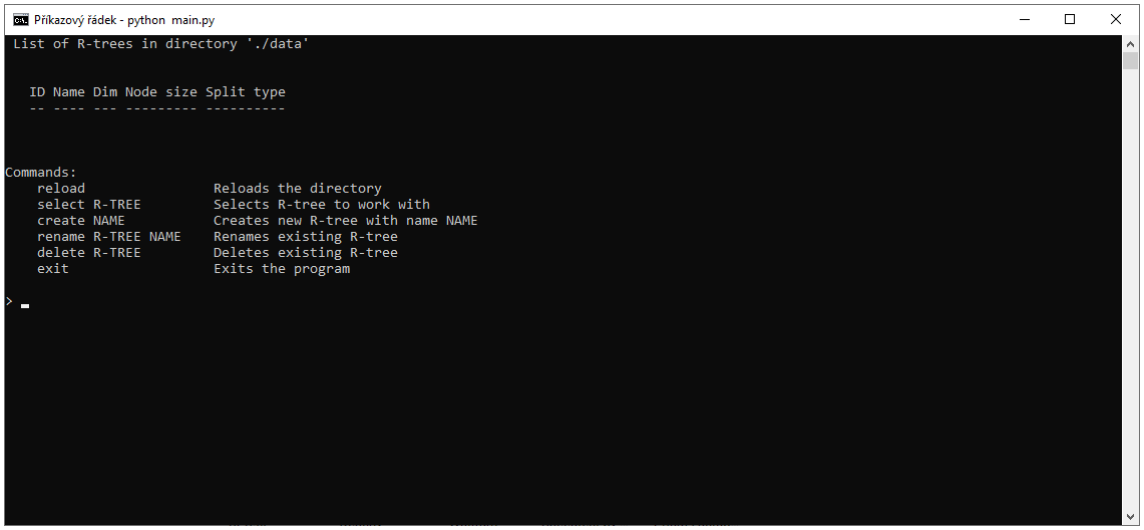
R-strom je možné vytvořit in memory, nebo v souboru. Třída zajišťující serializaci R-stromu na disk má svou vnitřní cache, o konstantní velikosti, ve které si drží uzly. Vždy pokud si R-strom zažádá o uzel s nějakým identifikátorem, třída se podívá do cache na indexu id uzlu modulo velikost cache. Pokud se na této pozici v cache nachází správný uzel, třída ho vrátí, pokud se na pozici nenachází žádný uzel, třída načte data z disku a pokud se na pozici nachází jiný uzel, třída nejdříve zkontroluje, zda se data uzlu změnila od těch která jsou v souboru, zjistí to díky flagu, který si u každého záznamu udržuje, a pokud ano, tak je nejprve uloží a až poté načte požadovaný uzel. Obdobně to funguje pokud R-strom chce uzel na nějakém id přepsat novým uzlem.

# 3 Implementace

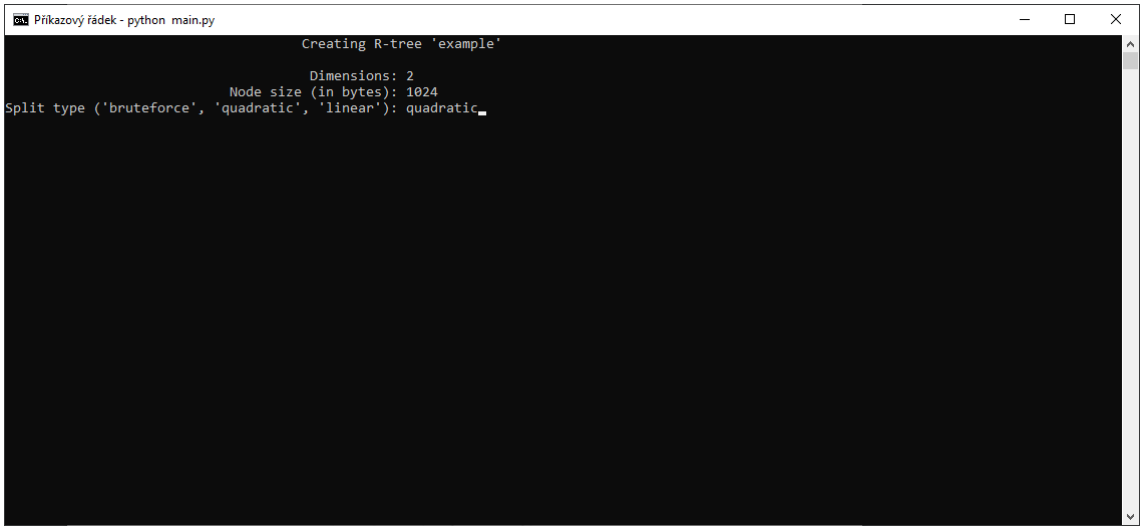
Celý projekt byl naprogramovaný v jazyce Python, konkrétně na verzi 3.9.2. Pravděpodobně bude projekt fungovat i na starší verzi, ovšem nebyl na ní testován. K vizualizaci R-stromu jsme využili knihovnu plotly verze 4.14.3. Hlavní aplikace je postavená, jako virtuální konzole, kde se jednoduchými příkazy dají nad R-stromy provádět různé operace, například vytvoření nového, smazání, přejmenování, vložení zadaného počtu náhodných dat atd.

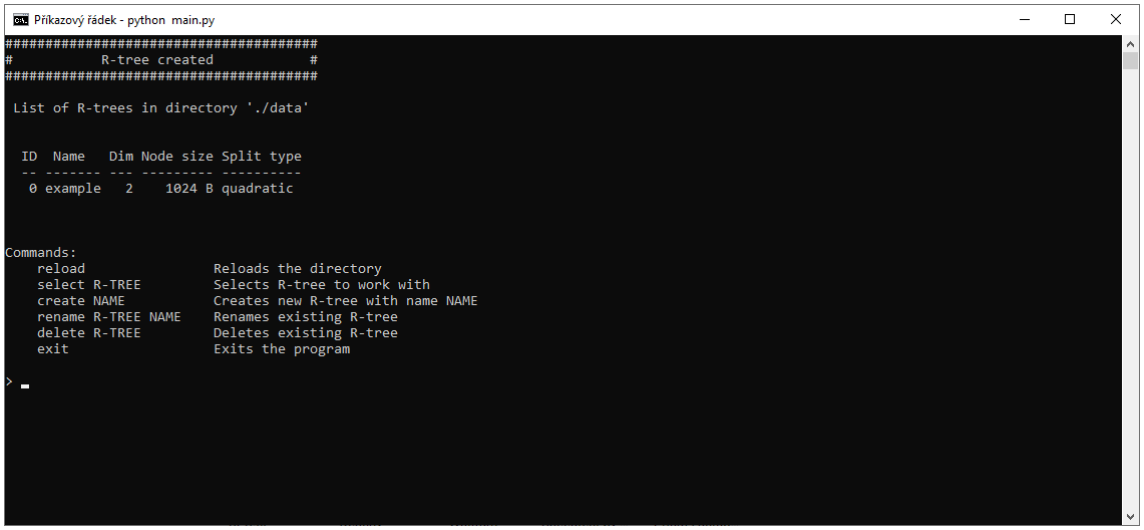
# 4 Příklad výstupu

Při prvním spuštění by výstup hlavní aplikace měl vypadat následovně.

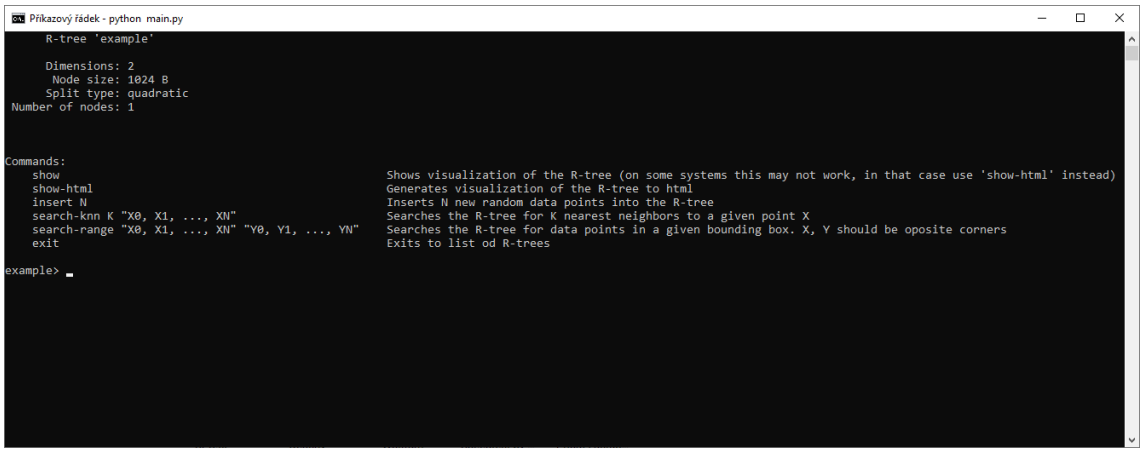



Po spuštění příkazu create example se nám zobrazí dialog, který nám dovolí nastavit parametry vytvářeného R-stromu.

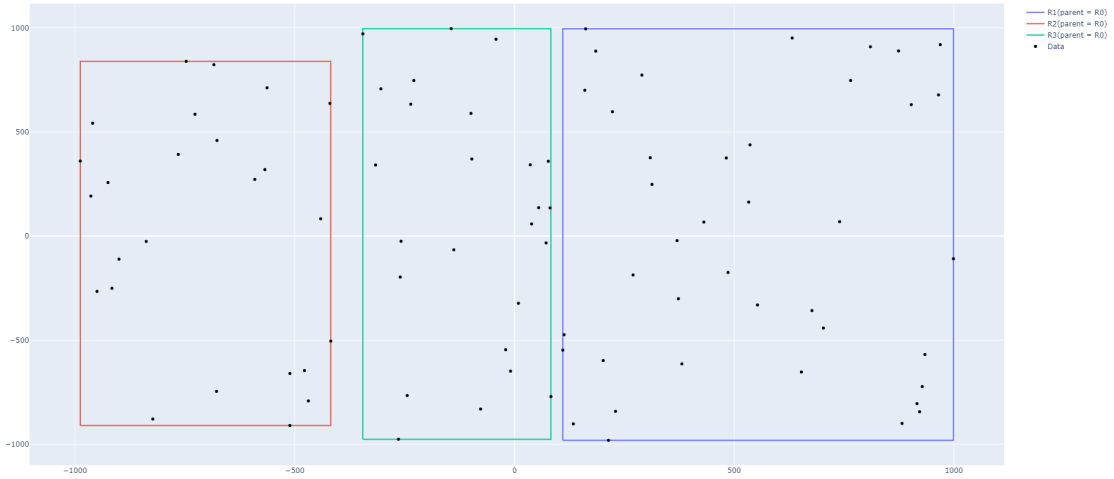


Příkazem select example si můžeme zobrazit detail R-stromu. V tomto detailu lze poté do stromu přidávat data a provádět nad ním dotazy.



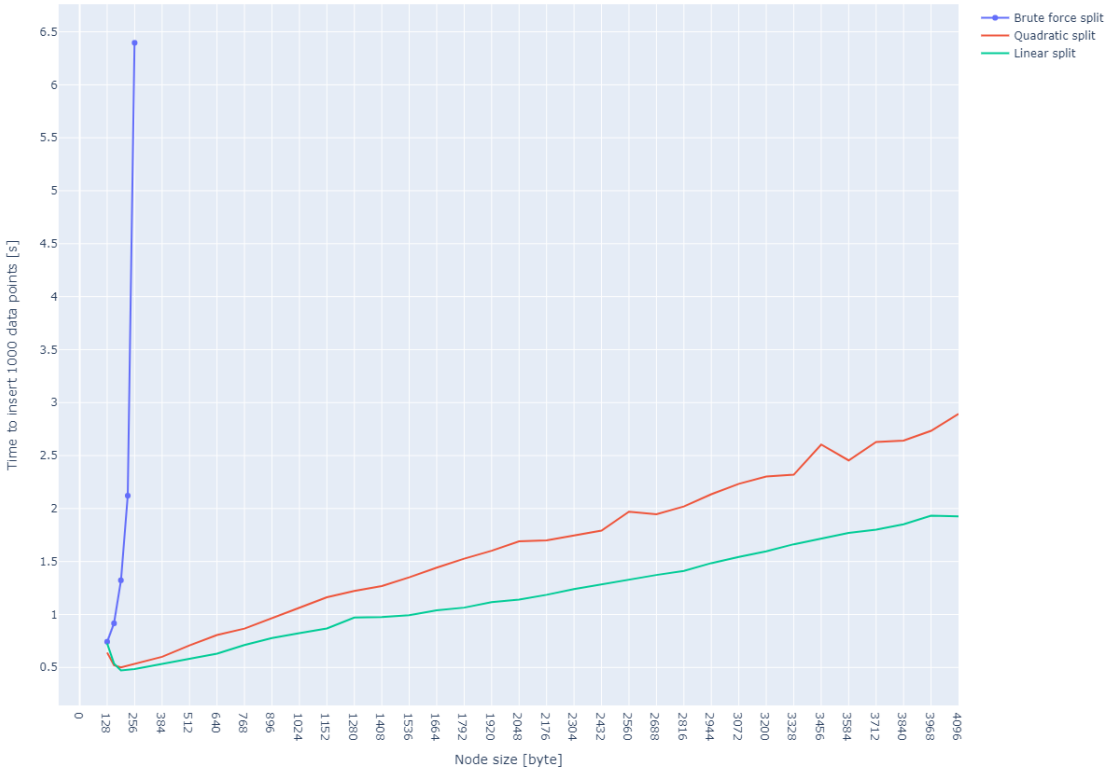

Zároveň si lze zobrazit vizualizaci R-stromu, která může vypadat například takto.

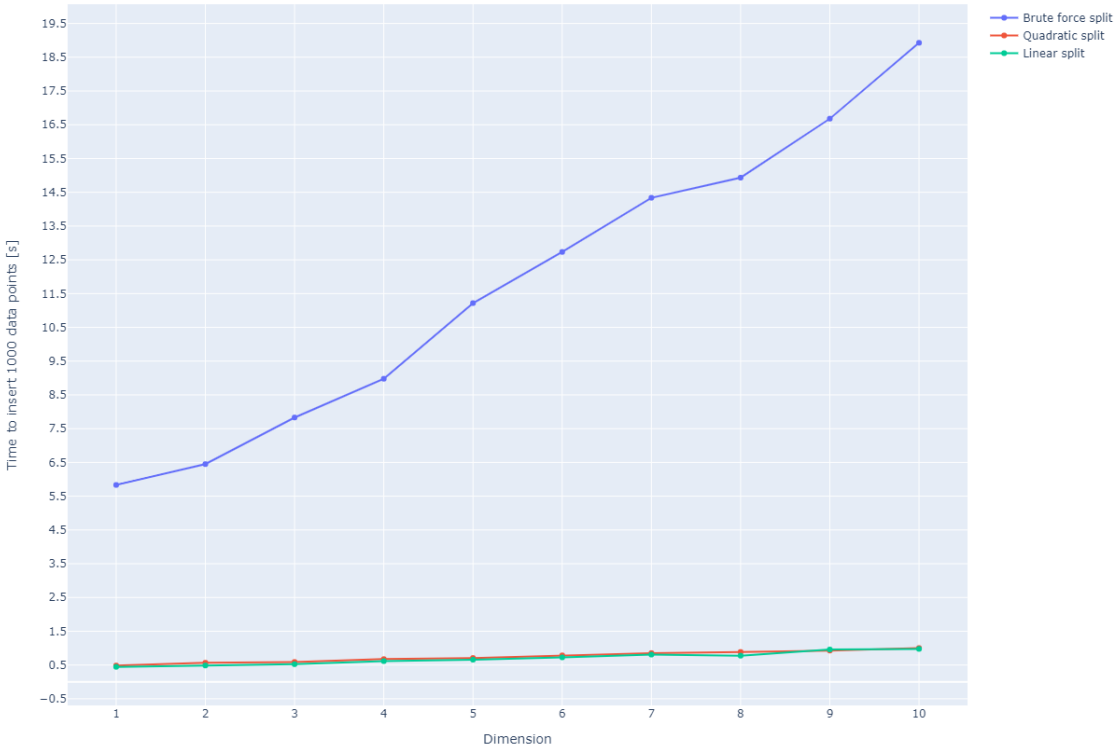


# 5 Experimentální sekce

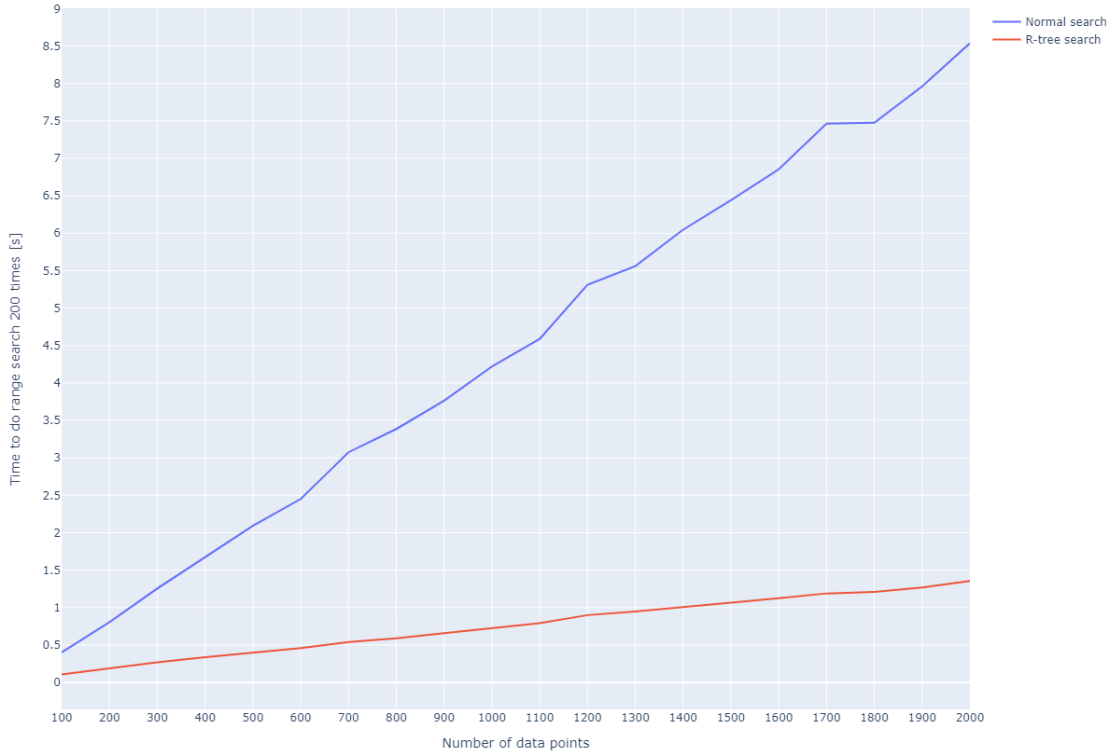
U R-stromu se dá testovat rychlost vkládání a rychlost hledání. Tyto hodnoty se mění, kromě klasických proměnných (počet záznamů), podle několika hlavních faktorů: velikost uzlu, algoritmu na rozdělení a dimenze záznamů. Na následujících grafech je vidět rozdíl efektivit různých dělících algoritmů podle dimenze a velikosti uzlu a to jak vkládání, tak hledání. Algoritmus dělení hroubou silou musel být testován jen na datech s malou velikostí uzlu, kvůli své rychlosti $O(2^n)$.

Z následujících grafů je zřejmé, že v rychlosti vkládání je opravdu nejrychlejší lineární dělící algoritmus a výrazně nejpomalejší algoritmus dělení hrubou silou.

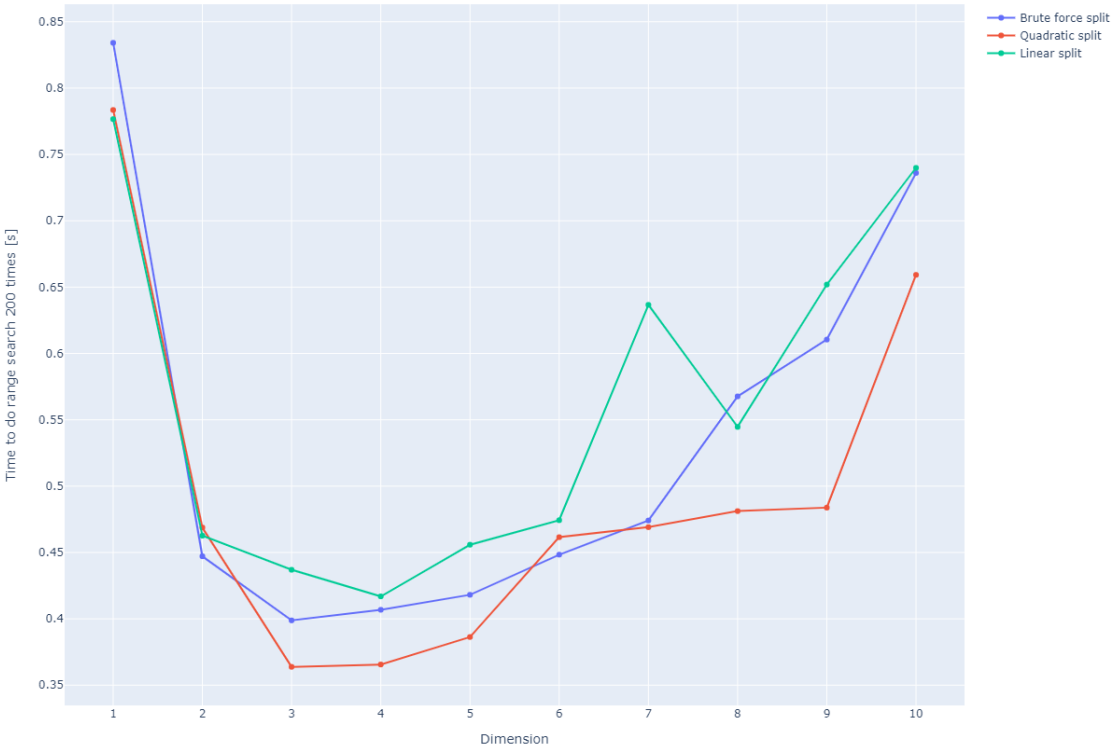


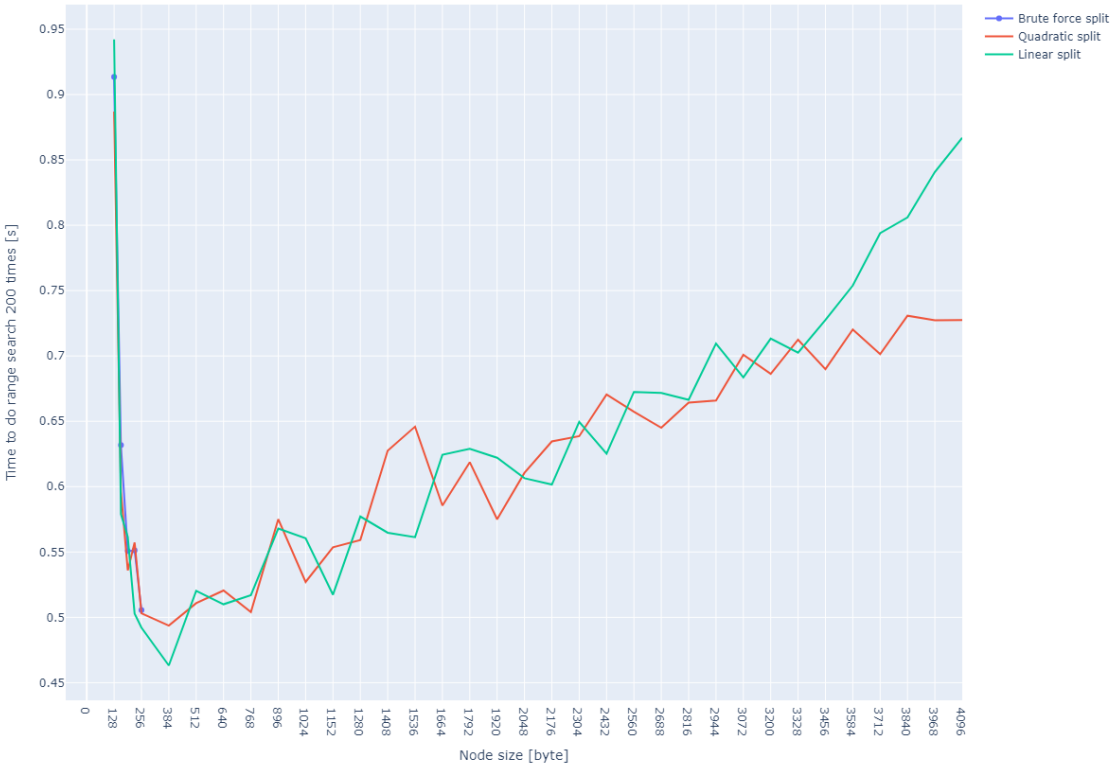


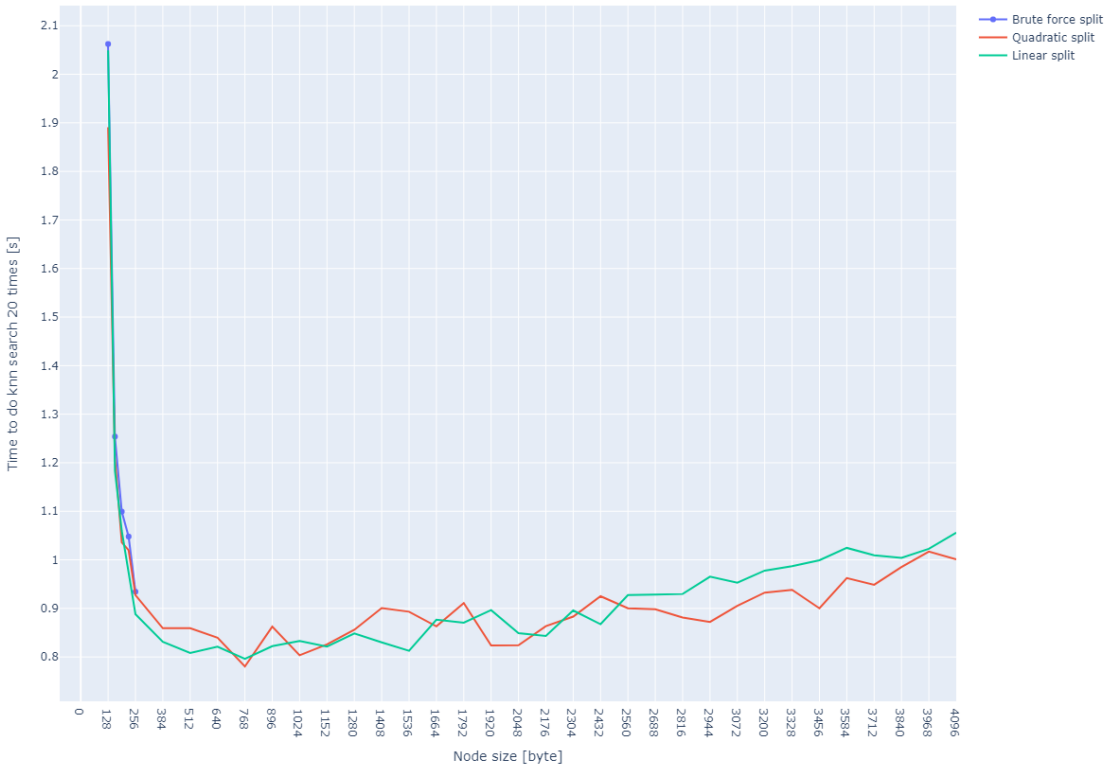

Na následujícím grafu je znázorněna efektivita hledání oblasti v R-stromu oproti sekvenčnímu hledání.

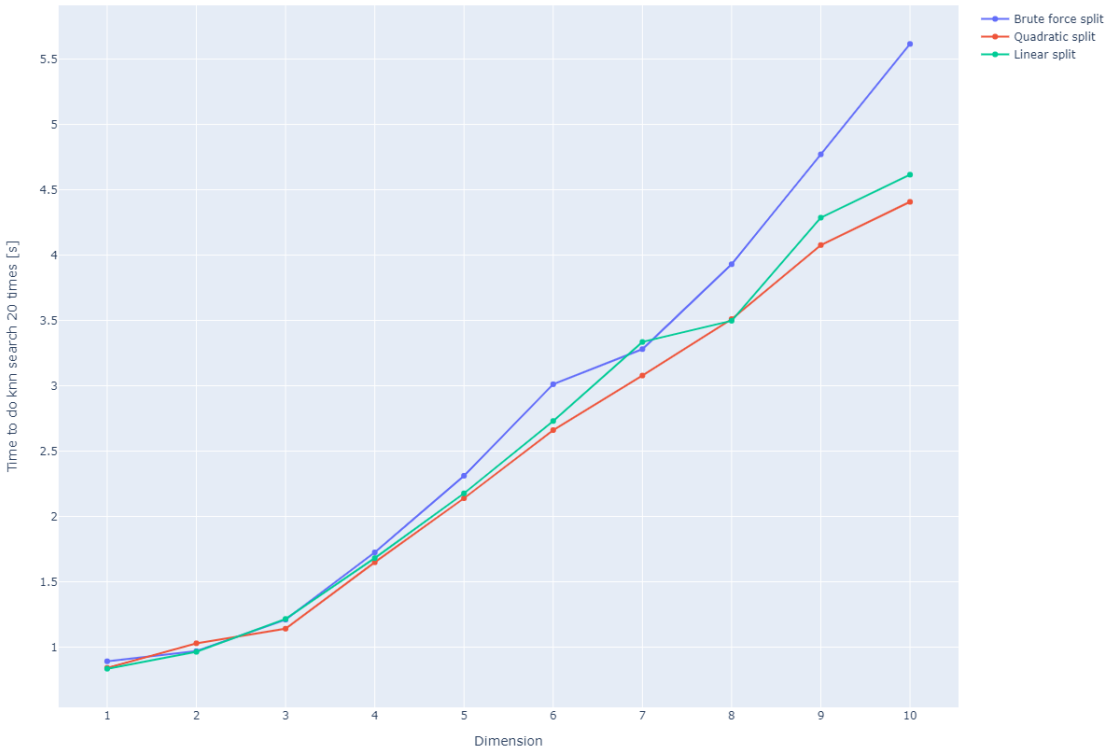



Zbytek grafů ukazuje efektivitu hledání podle použitých dělících algoritmů.


Brute force split
Quadratic split
Linear split
Time to do range search 200 times [s]
Dimension
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
1
2
3
4
5
6
7
8
9
10


Brute force split
Quadratic split
Linear split
Time to do range search 200 times [s]
Node size [byte]
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0
128
256
384
512
640
768
896
1024
1152
1280
1408
1536
1664
1792
1920
2048
2176
2304
2432
2560
2688
2816
2944
3072
3200
3328
3456
3584
3712
3840
3968
4096




Z celkového testování je jasně vidět že algortimus brute force je krajně nepraktický a od cca 10 záznamů v uzlu je vkládání již velice pomalé. Na menších datech, na kterých jsme program testovali je hledání poměrně stejně rychlé, pokud bychom ale testovali na větších datech, tak se

jemně projeví efektivita dělících algoritmů. Volbu algoritmu je tedy vhodné upravit podle frekvence vyhledávání a vkládání.

Z ohledu velikosti uzlu ze všech testů jasně vychází nejlépe počet záznamů někde okolo 30 - 50 záznamů na uzel. Při větším počtu záznamů na uzel strom ztrácí efektivitu protože pro každý uzel je třeba procházet lineárně větší počet záznamů. Tomu se chceme vyvarovat. Pokud se ale počet záznamů na uzel nastaví moc malý, tak je třeba výrazně frekventovaněji volat dělící algoritmy, které jsou ze své podstaty zpomalení.

# 6 Diskuze

V našem projektu jsme, po provedení experimentů zjistili, že naivní řešení kNN hledání které jsme implementovali, je pomalejší než sekvenční vyhledávání. Náš algoritmus začíná se vzdáleností nejvzdálenějšího bodu od hledaných souřadnic a pro tuto vzdálenost najde všechny body, které se v ní nachází. Hledání v dané vzálenosti probíhá obdobným algoritmem, jako u hledání v rozmezí, jen s upravenou podmínkou protnutí dvou boxů. Pokud algoritmus nalezne moc bodů, zmenší se vzdálenost na polovinu a naopak. Tento algoritmus je ale neefektivní, jelikož od začátku nevíme v jaké vzdálenosti musíme hledat. Obecně se tedy hledání všech prvků v určité vzdálenosti musí provést několikrát a délka běhu algoritmu nakonec překročí délku běhu sekvenčního řešení. Na hledání KNN nějakých souřadnic v R-stromu existují i rychlejší algoritmy, které například pracují s prioritní frontou. Jelikož se ale jedná o proof-of-concenpt projekt, rozhodli jsme se nakonec pro tento algoritmus.

# 7 Závěr

V teto semestrální práci jsme si ověřili výhody R-stromu pro ukládání n-dimenzionálních objektů. Také jsme prozkoumali výhody a neýhody různých typů dělích algoritmů a jejich vliv na běh a funkčnost R-stromu.