A picture containing diagram

Description automatically generated**ŽILINSKÁ UNIVERZITA V ŽILINE**

**FAKULTA RIADENIA A INFORMATIKY**

**Algoritmy a Údajové Štruktúry 2**

1. Semestrálna práca

Juraj Jánošík 5ZIS22 2023/2024

Dokumentácia údajové štruktúry sem1.

# Návrh systému Obrázok, na ktorom je diagram, rad, vývoj, text Automaticky generovaný popis

Nehnuteľnosť obsahuje ArrayList Parciel kde ak leží daná nehnuteľnosť na danej parcele sa priradí a naopak Parcele sa priradí nehnuteľnosť že v nej leží.

# Popis štruktúry programu:

Môj program sa skladá z triedy „Main“ z ktorej sa aplikácia spúšťa, ďalej obsahuje triedy ktoré sa zaoberajú chodom programu, medzi tie patria „GPS, IData, LandShapes, Node, Property, QuadTree, Shape“. Na otestovanie som si vytvoril triedu Tester kde testujem ako funguje môj program. Pre vykreslenie som si vytvoril triedy GUI v ktorej vytváram vizuálne hľadisko programu a triedu Controller v ktorej logika medzi buttonami a stromom.

GPS -> trieda má 4 atributy – x,y,horizontal,vertical.

IData -> je to interface ktorý využívam pri LandShapes, mám tam metódy CompareTo, canCreateSons, canCreateSonsOptimize, calculateX a calculateY.

LandShapes -> abstraktná trieda ktorá dedí a implementuje IDATA, využívam ju na parcely a nehnuteľnosti ako triedu z ktorej dedia compareTo aby som nemal duplicitný kód pre rovnaké triedy obsauje – GPS first,second , notes, identifier.

Property a Shape -> Snažil som sa všetko robiť po anglicky celý projekt nazval som Property ako nehnuteľnost a Shape ako parcelu. Obsahujú Property arraylist Shapov a Shapy arraylist Propertov.

Node-> je to list stromu obsahuje x1,y1,x2,y2 , depth , arraylist data a 4 svojich synov a iné metódy na pomoc s operáciami v strome.

QuadTree -> trieda obsahuje funkcie ako Insert, Delete, Search, Health, ChangeDepth, Optimize a OptimizeInsert, a pár pomocných privatných tried: countLandsInMaxDepth, countFreeNodes, countLands, findAllLands, findNode, checkIfNodeIn

## Optimalizácia:

Optimalizáciu som navrhol tak, že pri inserte nastane zmena a to taká že sa nebude root deliť na polovicu ale budú vznikať iné útvary ako štvorce, ak už má root synov potom delím tých synov na polovicu dal som tam algoritmus ktorý funguje nejako takto : Obrázok, na ktorom je rad, diagram, snímka obrazovky, štvorec

Automaticky generovaný popis

Majme strom v ktorom už je prvok ktorý je veľký a je v roote nedokáže ísť do syna (nie je na obrázku vyššie). Pridávam prvok ktorý ma by mal ísť povodne do stromu bez optimalizácie do 1. syna na severo-západ. V mojej optimalizácií to funguje tak že sa vypočíta stred stromu a z neho sa môže posunúť vývoj ďalších synov maximálne o 1/5 pôvodného nodu inak povedané môže sa posunú maximálne o 1/5 celej dĺžky od stredu. Ak nový prvok má nejakú hranu v štvorci zo stredu tak nový stred sa vypočíta na hranu daného prvku a vytvoria sa synovia ak nemá tak sa dá stred. Takto som vyriešil to že ak by tam bol nejaký prvok moc malý a bol by napríklad úplne dolu vľavo aby nevznikol syn taký malý že je skoro žiadna šanca že sa tam niečo zmestí. Po vykonaní optimalizácie ešte kontrolujem či by nebolo dobré aj zmeniť výšku stromu. Napríklad ak tam mám málo prvkov a mám 2 synov vedľa seba ktorý sú malý tak sa mi vytvorí toľko nodov koľko je výška stromu to som dal do podmienky nech sa upraví výška stromu. Taktiež mám opačný prípad ak je veľa prvok v listoch stromu a bráni im pri vytváraní výška a je dosť pvkov v strome tak zvýšim výšku stromu dal som to na 3 odmocninu zo všetkých prvkov ktoré sa nachádzajú v strome.

# Insert

Insert obsahuje parameter T data. Na začiatku si vytvorím Node<T> do ktorého vložím root. Vytvorím si pomocné zásobníky nazval som jeden queue a druhy newData. Metóda ide kým je nejaký prvok v zásobniku queue. Vytvorím si ešte premennú T dataQueue čo bude naznačovať prvok ktorý aktuálne riešim. Na začiatku ak je dataQueue null popnem prvok zo zásobníka queue. Pomocou compareTo rozhodnem kam má ísť. Postupne idem hlbšie do stromu ak sa prvok ešte nepriradil a v prípade že ich je viac ktoré treba priradiť dávam ich do queue. Ak náhodou nie sú synovia vytvorený a môžu sa vytvoriť, tak ich vytvorím. Druhý zásobník využívam v prípade keď daný node má viac dát tak skontrolujem či môžu ísť dam ich do queue a nemôžu dám ich do newData potom clearnem arraylist a nahádžem tie data z newData nanovo.

# Delete

Delete obsahuje parameter T data.Na začiatku si vytvorím zasa Node<T> node kde dám root. Využívam Stack s názvom queue. Na začiatku si musím nájsť prvok ktorý chcem zmazať, node v ktorom sa nachádza uložím do node pomocou metódy findNode. Ak ho nájdem postupne pokračujem. Hneď ho zmažem a skontrolujem či daný node treba upravovať. Potom skontrolujem či má synov alebo nemá a pomocou cyklu kontrolujem smerom do hora stromu či sa môže mazať. Pokračujem kým sa nemôže zmazať a potom skončím.

# TreeHealth

Metóda nemá parameter. Využíva viaceré privátne metódy aby som dokázal vypočítať zdravie stromu. Je to typu int a vracia mi (počet nodov s dátami – počet nodov s dátami ktoré majú viac dát ako priemerný node). Priemer si vypočítam pomocou všetkých prvkov v strome / počet nodov vytvorených v strome. Prechádzam celý strom a ak nájde node kde sú prvky a majú viac ako priemer spočítam si ich a nakoniec to odčítam. Ten strom čo má toto číslo **väčšie** tak je pre mňa optimálnejší.

# Search

Metóda obsahuje 4 parametre: x1,y1,x2,y2. Je typu ArrayList<T>. Postupne prechádzam pomocou comparatora prvky a idem hlbšie do stromu. Prvky si ukladám do zásobníka a po každom cykle prebehnem všetky prvky a pomocou compareTo to porovnám či pretínajú danú oblasť ak áno priradím ich do arraylist. Cyklus ide pokým je v zásobniku viac ako 1 prvok.

# ChangeDepthTree

Metóda má jeden parameter a to newDepth. Na začiatku si priradím Node a postupne si ukladám prvky do zásobniku a idem hlbšie do stromu. Ak hĺbka sa má zväčšiť potom nájdem všetky prvky ktoré sa nachádzajú na starej hĺbke a nanovo ich insertujem a zmažem. Ak sa má hlbka zmenšiť postupne prechádzam od najnižšie syna až po novú výšku prvky a dám ich do otca. Problém bol insert ale vyriešil som ho tak že zmením root na daný node. Starý root si uložím a nakoniec root zasa zmením na starý root. Na konci tiež zmením výšku.

# Meranie rýchlosti na 10000 prvkoch pri 50000 opakovaní

## QuadTree INSERT:

Obrázok, na ktorom je text, snímka obrazovky, písmo, dokument

Automaticky generovaný popis

Na obrázku je vidieť 1212 ms čo pre nás znamená po vydelení 50000 - > **24,24** **mikrosekúnd** je priemerná dĺžka operácie insert s 10000prvkami

## QuadTree DELETE:

Obrázok, na ktorom je text, snímka obrazovky, písmo, číslo

Automaticky generovaný popis

Na obrázku je vidieť 35 183ms čo pre nás znamená po vydelení 50000 - > **703,66** **mikrosekúnd** je priemerná dĺžka operácie delete s 10000prvkami

## QuadTree OPTIMIZE INSERT:

Obrázok, na ktorom je text, snímka obrazovky, písmo, dokument

Automaticky generovaný popis

Pre rapídne zhoršenie som to nerobil na viac ako 10000 replikáciach priemerný čas je preto 133s/10000= **13,3 milisekúnd** na jednu operáciu. Nepodarilo sa mi to optimalizovať i keď štruktúra sa javí ako zdravšia. Menší problém bol v tom že keď som pustil štruktúru cez profiler rapídne narástol čas vykonávania.

## QuadTree OPTIMIZE INSERT:

Obrázok, na ktorom je text, snímka obrazovky, písmo, číslo

Automaticky generovaný popis

Pri delete mi vyšlo 4057 ms čo je v prepočte **405,7 mikrosekúnd** priemerne na jednú operáciu čiže ide o zlepšenie oproti normálnej štruktúre takmer o 300 mikrosekúnd

## Užívateľská príručka:

Na začiatku v hornom rohu sú 3 okná na dlžku šírku a hĺbku stromu, ak idem vytvárať nový strom je nutné ich vytvoriť. Pod nimi sú X a Y okná pre vytvorenie súradníc. Pod nimi je priestor na vytvorenie buď shapu alebo property taktiež sa dá editovať, hľadať alebo mazať. Je nutné vyplniť GPS body a to tak že si ich vypíšete pomocou tlačítka ShowAllGPSLocations a číslo vľavo reprezentuje danú GPS súradnicu. Pri načítaní treba dať buď 1 alebo 2 do INDEX land shape čo značí shape alebo property a stlačiť load alebo save. Pre optimalizáciu treba dať 1 do indexu keď chcete shape alebo 2 ak chcete property. V prípade že chcete vedieť zdravie optimalizovaného stromu v prípade shapov napíšete 2 do indexu a dáte TreeShapeHealth, 1 ak chcete bez optimalizácie to isté platí pre property. Reset zresetuje všetko na obrazovke. Pri zmene hĺbky treba dať newDepth a kliknúť shape alebo property záleží čo potrebujete. V prípade že sa vám nechce vytvárať nové shapy a chcete ich náhodne môžete dať do index 1 alebo 2 v prípade že chcete 1 neoptimalizovaný a 2 optimalizovaný strom naplniť a do insertu 100 delete 0 counter je koľko chcete premenných a dáte buď testAllShape alebo testAllProperty podľa toho čo chcete naplniť. Test Insert funguje tak že musíte mať prázdny strom a dáte počet prvkov do counter, funguje len na shape. Pri test search musí byť taktiež prazdny funguje len na shape vytvoríte shape potom dáte counter koľko prvkov má naplniť a potom kliknete test search. Test speed funguje tak že najprv naplníte buď z loadu alebo cez allShape alebo property. Bude treba zadať index či chcete shape(1) property(2) a vpravo je checkbox či chcete optimalizovaný alebo nie insert a počet prvkov. Tlačítko optimize funguje tak že máte naplnený nejaký strom a dáte buď 1 alebo 2 do indexu podla toho či chcete shape(1) alebo property(2).

UML

Shape+Property+IData+Landshape+GPS  
Obrázok, na ktorom je text, diagram, číslo, písmo

Automaticky generovaný popis

QuadTree Obrázok, na ktorom je text, snímka obrazovky, diagram, písmo

Automaticky generovaný popis

Node

Obrázok, na ktorom je text, snímka obrazovky, diagram, rad

Automaticky generovaný popis