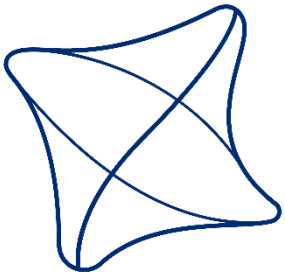
Žilinská univerzita v Žiline
Fakulta riadenia a informatiky



Semestrálna práca

Algoritmy a údajové štruktúry 1

Dominik Ježík

2023

1 Návrh aplikácie

Aplikácia sa skladá z viacerých tried, pričom hlavnou triedou je trieda *App*, z ktorej vytvoríme inštanciu vo vstupnom bode programu (vo funkcií main). Táto trieda obsahuje v atribútoch údajové štruktúry určené na uchovávanie územných jednotiek podľa jednotlivých úrovni zadania semestrálnej práce.

Po zavolaní metódy *start* sa spustí program aplikácie. Najskôr inicializujeme používané údajové štruktúry a vytvoríme inštanciu triedy *DataUnitLoader*, pomocou ktorej načítame vstupné dáta z príslušných súborov. Kódovanie súborov bolo upravené na Windows-1250 pre jednoduchšiu prácu so slovenskou diakritikou a následným vypisovaním na konzolu.

Používateľ dokáže s aplikáciou komunikovať prostredníctvom konzolového rozhrania. Na výpis a vstup slúži trieda *Console*, ktorá obsahuje metódy pre výpis (aktuálneho menu, výzvy používateľovi na zadanie vstupu, výpis obsahu údajovej štruktúry, rámčeky aplikácie a podobne).

1.1 Reprezentácia územných jednotiek

Územné jednotky zdieľajú rovnaký typ údajov, preto je v programe predok každej územnej jednotky reprezentovaný triedou *DataUnit*. Túto triedu rozširuje trieda *Obec*, ktorá navyše pridáva atribút *population_density*, čiže hustotu obyvateľstva na kilometer štvorcový (bonusová úroveň).

Triedu *DataUnit* tiež rozširuje trieda *AggregatedDataUnit* reprezentujúca vyššiu územnú jednotku, ktorá navyše poskytuje agregovanú informáciu o priemernej hustote obyvateľstva na obec. Túto triedu rozširujú triedy *Kraj* a *Okres*.

Vo viacerých miestach aplikácie používame dynamic cast, z toho dôvodu potomkovia triedy DataUnit prekrývajú metódu *dummy*, ktorá nevykoná žiadny kód. Slúži iba na to aby sme mohli dynamic cast používať.

1.2 Načítanie vstupných údajov do aplikácie

Na načítanie vstupných údajov slúžia triedy *DataUnitLoader* a *CsvLoader*. *CsvLoader* obsahuje metódu *load*, ktorá otvorí súbor zo zadanej cesty a pre každý riadok csv súboru spustí zadanú funkciu, kde ako parameter pošle dynamicky alokované pole segmentov čítaného riadku.

DataUnitLoader v konštruktore triedy získa cesty k súborom s údajmi a ukazovateľ na hierarchiu, ktorú inicializuje (pozri 6.1 Spôsob načítavania údajov do hierarchie). Táto trieda obsahuje metódy pre

načítanie každého typu územnej jednotky. Do parametra každej metódy posielame funkciu, ktorá sa spustí pre každú jednu územnú jednotku, pričom poskytne územnú jednotku inicializovanú ako dynamicky alokovaný objekt svojho typu (*Kraj*, *Okres*, *Obec*) prostredníctvom parametra. Metóda vždy volá metódu z inštancie *CsvLoader*, odkiaľ získa príslušné dáta.

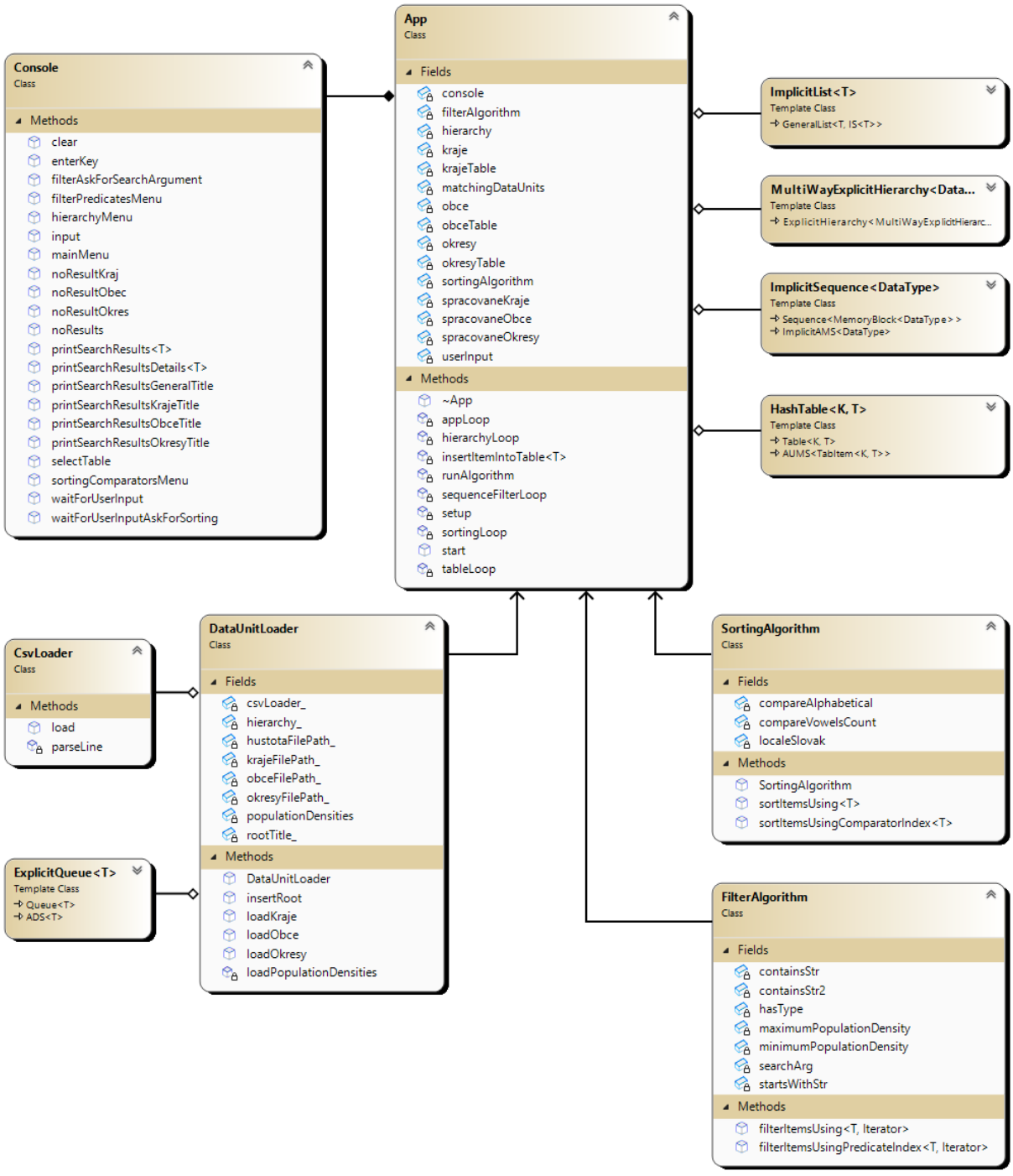
V triede *App* si takto inicializujeme údajové štruktúry príslušnými územnými jednotkami. V deštruktore triedy tieto objekty musíme z haldy uvoľniť aby nevznikli memory leaky.

1.3 FilterAlgorithm a SortingAlgorithm

Trieda *App* ešte využíva inštancie tried na filtrovanie a zoradenie územných jednotiek. Detaily o týchto triedach sú vysvetlené v implementáciách jednotlivých úrovní semestrálnej práce.

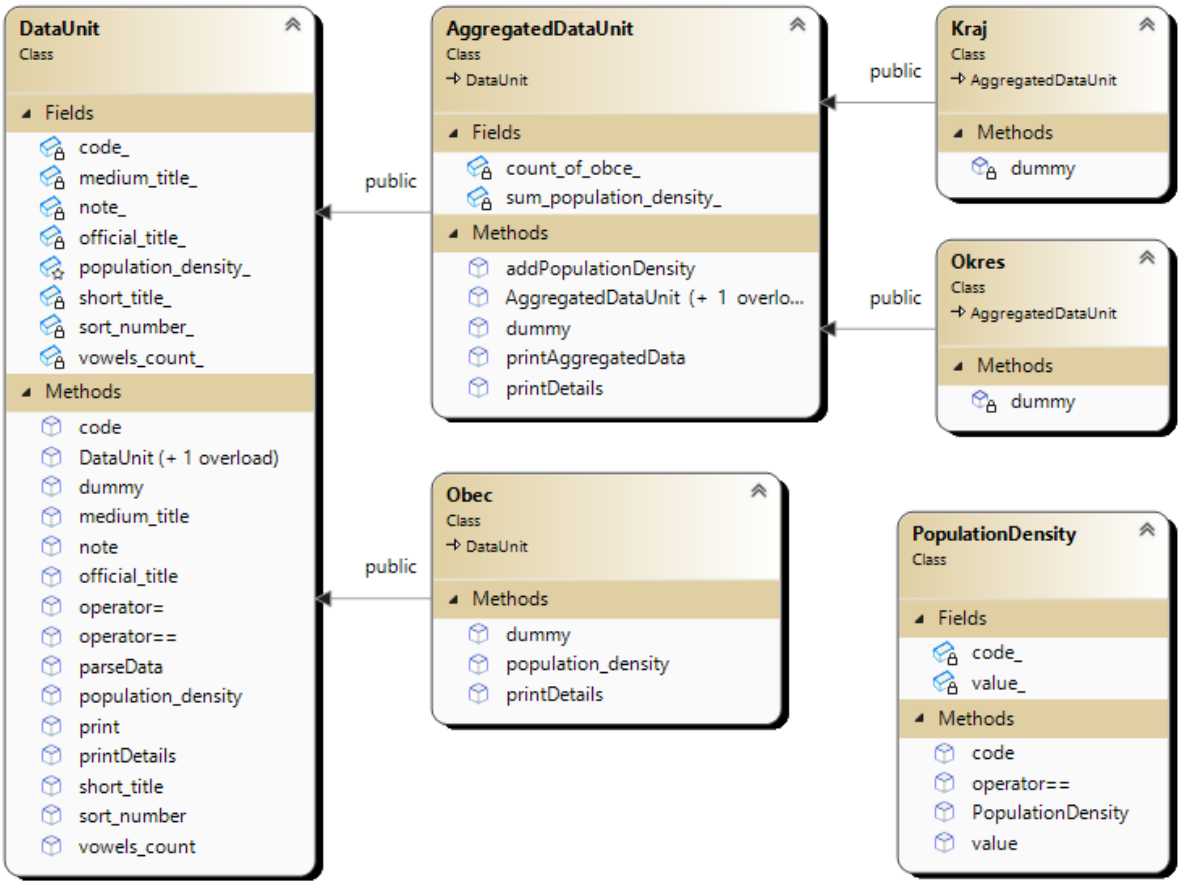
2 UML Diagramy tried aplikácie

2.1 Hlavné triedy aplikácie



Obr. 1 UML diagram hlavných tried aplikácie (vnútorný pohľad)

2.2 Územné jednotky a hustota obyvateľstva



Obr. 2 UML diagram tried predstavujúcich dáta aplikácie (vnútorný pohľad)

3 Rozbor vhodnosti použitia zvolených údajových štruktúr

3.1 Prvá úroveň

Na načítanie a uchovanie územných jednotiek sú použité tri implicitné zoznamy (*ImplicitList*), pre každý typ územnej jednotky jeden. Do zoznamov nie sú okrem prvotného načítania pridávané ďalšie územné jednotky, a preto nehrozí expanzia po dosiahnutí limitu štruktúry. Zoznam tiež počas behu programu nijako nemodifikujeme. Prístup k ľubovoľnému prvku je možný so zložitosťou O(1), preto je časovo efektívny. Neexistuje tu vzťahová časť, preto je táto štruktúra pamäťovo efektívna.

Po spustení filtrovacieho algoritmu, územné jednotky ukladáme do ďalších troch implicitných zoznamov podľa typu jednotky. Tieto zoznamy následne používame na výpis, kde je prístup k položkám časovo efektívny.

3.2 Druhá úroveň a štvrtá úroveň

Na načítanie územných jednotiek do hierarchie používame viaccestnú explicitnú hierarchiu (*MultiWayExplicitHierarchy*). V tomto prípade nemáme na výber iný typ hierarchie, pretože počet synov každého vrcholu je rôzny. V tom prípade nemôžme použiť K-cestnú hierarchiu, pretože by sme zbytočne plytvali pamäťou.

Univerzálny algoritmus z prvej úrovne je možné použiť aj nad hierarchiou a jej výsledky ukladáme do jednej implicitnej sekvencie (*ImplicitSequence*). Táto sekvencia bola zvolená z dôvodu, že je používaná v štvrtej úrovni na utriedenie územných jednotiek. Aby sme mohli efektívne vykonať triedenie potrebujeme používať implicitnú pamäťovú štruktúru, kvôli efektívnemu náhodnému prístupu.

3.3 Tretia úroveň

V tejto úrovni chceme efektívne pristupovať k územným jednotkám na základe ich názvu. Zvolili sme tri tabuľky s rozptýlenými záznamami (*HashTable*) pre každý typ územnej jednotky jednu. Teoreticky ide o najrýchlejšiu tabuľku – operácie vlož a nájdi majú zložitosť O(1) a s vhodnou hashovacou funkciou sa k tejto zložitosti dokážeme priblížiť. Detaily o zvolených hashovacích funkcia a nameranej časovej zložitosti sú v 7 Implementácia tretej úrovne SP.

3.4 Bonusová úroveň

V bonusovej úrovni používame explicitný front (*ExplicitQueue*) na uloženie hustoty obyvateľstva. Spoliehame sa tu na poradie prvkov, preto je front vhodnou štruktúrou. Explicitný front je pamäťovo vhodnejší, keďže implicitný front využíva viac pamäte ako nevyhnutne aktuálne potrebuje.

4 Používateľská príručka

Aplikácia poskytuje konzolové rozhranie. Po zapnutí aplikácie sa zobrazí hlavné menu, v ktorom si používateľ môže zadaním príslušného čísla vybrať:

- filtrovanie územných jednotiek (SP úroveň 1)
- navigácia v územných jednotkách (SP úroveň 2 + 4)
- vyhľadávanie informácií podľa názvu (SP úroveň 3)
- ukončiť aplikáciu

4.1 Filtrovanie územných jednotiek

Po zvolení, aplikácia používateľovi vypíše zoznam kritérií, podľa ktorých používateľ chce územné jednotky filtrovať – stačí zadať číslo zvoleného kritéria a potvrdiť. Potom je používateľ vyzvaný na zadanie výrazu, podľa ktorého sa má filtrovať – napr. všetky územné jednotky, ktoré začínajú na výraz „Veľká“. Aplikácia následne používateľovi vypíše všetky územné jednotky spĺňajúce kritérium. Následne sa používateľ môže vrátiť do hlavného menu zadaním písmena x.

4.2 Navigácia v územných jednotkách

Používateľovi sa vypíše zoznam krajov slovenskej republiky s číslom. Ak zadá niektoré z čísel presunie sa na daný kraj. V danom kraji sa vypíšu informácie o kraji a rovnako zoznam okresov daného kraja s číslom. Tak isto to funguje pri výbere okresu a výpis obcí. Číslom 0 sa používateľ vie dostať na nadradenú územnú jednotku. V ľubovoľnej pozícií vie používateľ spustiť filtrovací algoritmus (možnosťou *) rovnako ako v predchádzajúcej sekcií. Filtrovací algoritmus sa vykoná nad územnou jednotkou ktorá je aktuálne zvolená a nad všetkými jej podradenými jednotkami.

Používateľ po spustení fitrovacieho algoritmu dodatočne dokáže výsledky zotriediť (možnosťou &) podľa zvoleného kritéria – abecedne alebo podľa počtu samohlások.

4.3 Vyhľadávanie informácií podľa názvu

Používateľ zadá typ územnej jednotky a jej oficiálny názov. Aplikácia nájde a vypíše všetky informácie o danej územnej jednotke. V prípade, že je územných jednotiek s rovnakým názvom viacero, vypíše aj tie.

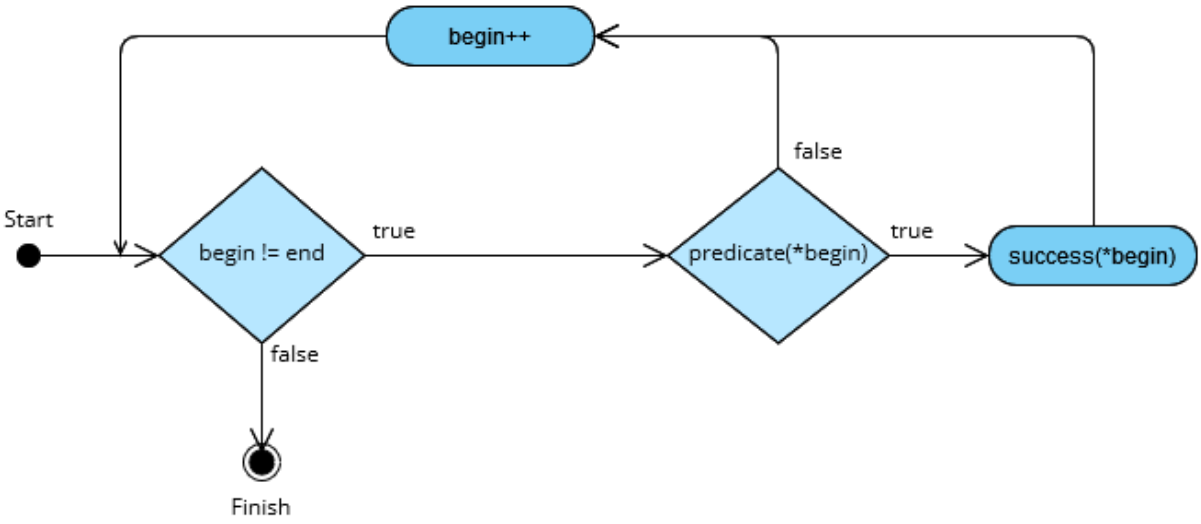
5 Implementácia prvej úrovne SP

Do zvolených sekvenčných údajových štruktúr (ImplicitList) sme načítali údaje spôsobom opísaným v sekcií Návrh aplikácie. Používateľa vyzveme na zvolenie predikátu, podľa ktorého si vyžaduje filtrovanie územných jednotiek a tiež ho vyzveme na zadanie vyhľadávacieho argumentu (napr. reťazec, na ktorý má územná jednotka začínať).

Za samotné filtrovanie je zodpovedná trieda *FilterAlgorithm*, ktorá obsahuje univerzálny algoritmus na filtrovanie. Vstupnými parametrami sú: pár iterátorov, predikát vo forme funkcie a funkcia, ktorá sa má vykonať keď filtrovaná položka vyhovuje predikátu.

Z triedy *App* ale používame inú metódu, kde nám stačí zadať zvolené číslo predikátu, vyhľadávací argument a tak ako pri univerzálnom algoritme pár iterátorov spolu s funkciou, ktorá sa má vykonať pre každú územnú jednotku vyhovujúcu zvolenému predikátu. *FilterAlgorithm* obsahuje ako atribúty 5 dostupných predikátov.

5.1 UML Diagram aktivít univerzálneho algoritmu



Obr. 3 Diagram aktivít univerzálneho algoritmu

5.2 Programátorská príručka

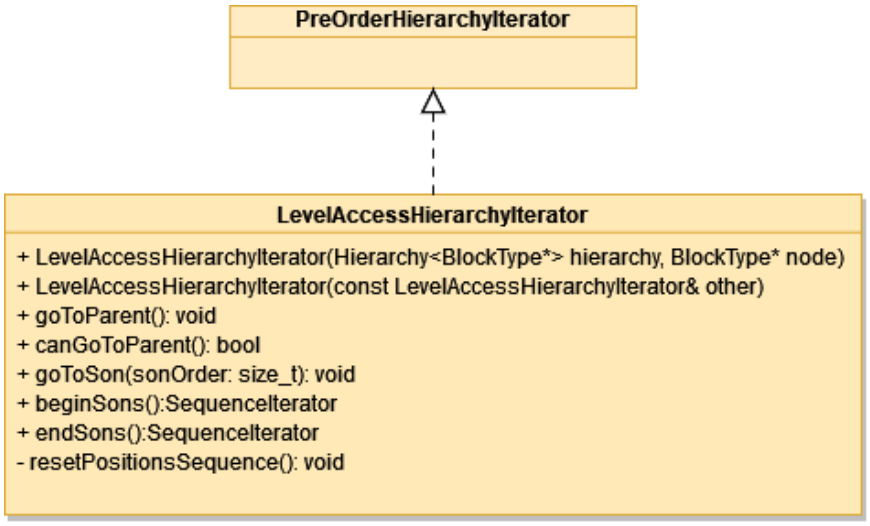
Aby sme mohli používať univerzálny algoritmus, musíme najskôr vytvoriť inštanciu triedy *FilterAlgorithm*. Algoritmus následne spustíme zavolaním metódy *filterItemsUsing*, kde do vstupných parametrov zadáme:

- pár iterátorov ukazujúci na začiatok a koniec položiek údajovej štruktúry (*begin*, *end*),
- predikát, ktorý je vo forme lambda funkcie alebo funkčného objektu,
- lambda funkciu alebo funkčný objekt, ktorý sa vykoná pre každú filtrovanú položku ak spĺňa zadaný predikát.

Metóda má dva šablónové parametre, vďaka ktorým je možné použiť iterátor ľubovoľného dátového typu a tiež filtrovať položky ľubovoľného typu.

6 Implementácia druhej úrovne SP

V druhej úrovni umožňujeme používateľovi pohyb v hierarchií územných jednotiek pomocou úrovňového iterátora (*LevelAccessHierarchyIterator*).



Obr. 4 UML diagram iterátora územných jednotiek

Tento iterátor sa nachádza v súbore *hierarchy.h* a rozširuje iterátor do hĺbky v priamom poradí (*PreOrderHierarchyIterator*). Umožňuje sa presunúť na otca, pokiaľ neukazuje na koreň hierarchie. Umožňuje sa presunúť na ľubovoľného syna aktuálneho vrcholu. Používateľ sa môže týmto iterátorom navigovať na ľubovoľnú územnú jednotku. Iterátor ukazujúci na ľubovoľný vrchol hierarchie tvorí jej podhierarchiu, pričom koreňom je tento vrchol. Aby sme boli schopný spustiť univerzálny algoritmus nad takouto podhierarchiou musíme pri každom volaní metód *goToParent* a *goToSon* vyčistiť sekvenciu pozícií. Túto sekvenciu si vytvára predok PreOrderHierarchyIterator, aby dokázal prechádzať celou hierarchiou. Tým že túto sekvenciu vyčistíme, pri prehliadke neopustíme našu podhierarchiu, pretože v sekvencií pozícií už nebudeme mať referenciu na vyššiu úroveň ako je koreň podhierarchie.

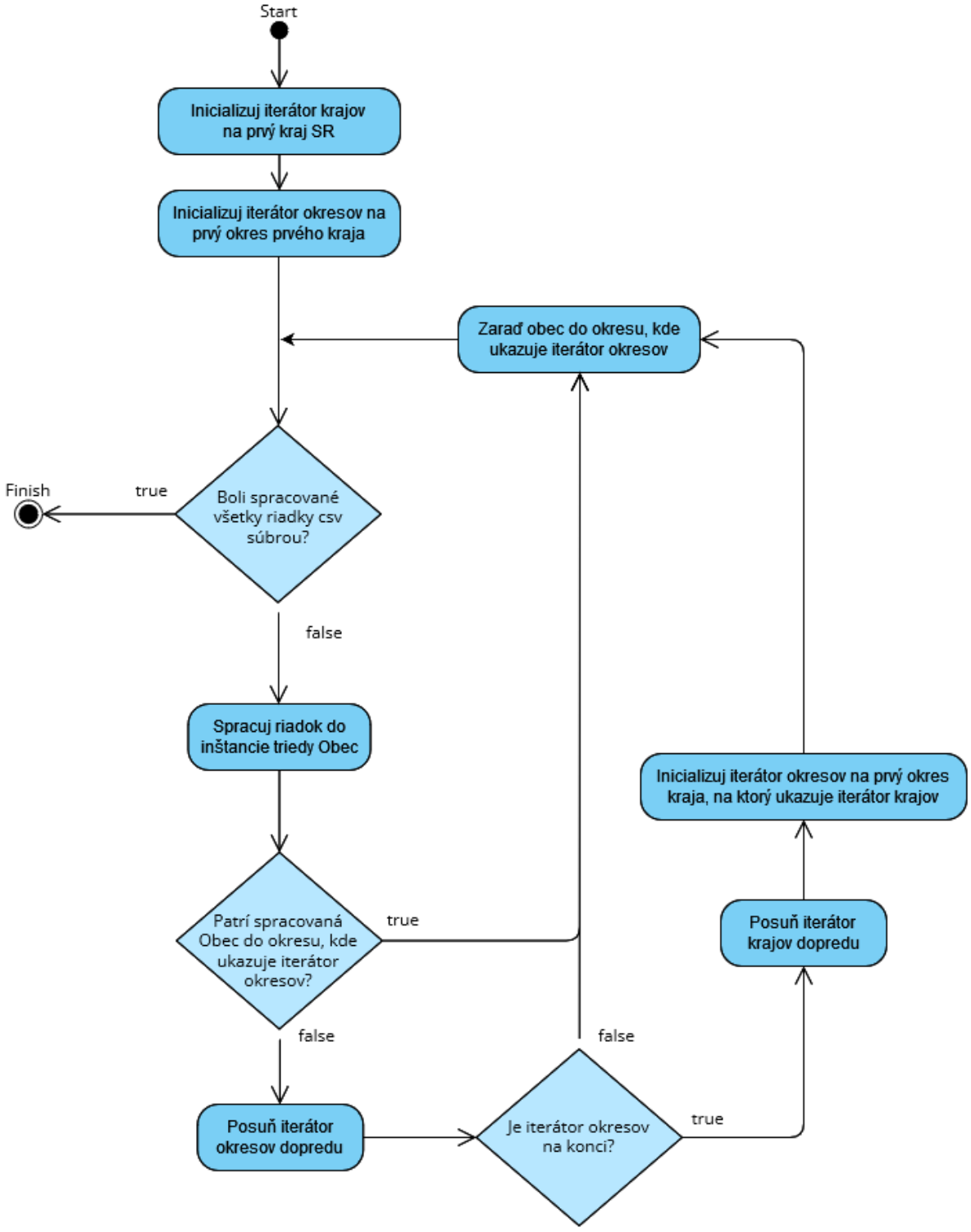
6.1 Spôsob načítavania údajov do hierarchie

Za načítavanie územných jednotiek do hierarchie je zodpovedná trieda *DataUnitLoader*. Na začiatku vloží do hierarchie ako koreň Slovenskú Republiku, ktorá je typu *AggregatedDataUnit* (keďže chceme aby poskytovala agregovaný výpis priemernej hustoty obyvateľstva).

Potom načítame do hierarchie kraje. Každý kraj spracujeme do inštancie triedy *Kraj* a vložíme medzi synov koreňa (Slovenská Republika) hierarchie. Zložitosť načítavanie je v tomto prípade O(n).

Pri načítaní okresov sa spoliehame na poradie údajov v csv súbore (okresy sú zoradené v rovnakom podarí v akom sú zoradené kraje, pod ktoré okresy patria) aby sme načítanie mohli vykonať efektívne. Na začiatku získame iterátor synov koreňa – čiže iterátor ukazujúci na prvý kraj. Pri každom spracovaní údajov do triedy *Okres* skontrolujeme, či daný okres patrí do kraju, na ktorý ukazuje iterátor (podľa kódu kraja a okresu). Ak zistíme, že okres do daného kraja nepatrí, posunieme iterátor krajov dopredu na ďalší kraj. Keďže sa spoliehame na poradie okresov a na skutočnosť, že v každom kraji je aspoň jeden okres, tak okres priradíme do správneho kraja. Toto zvolené načítavanie má zložitosť O(n).

Načítanie obcí funguje podobným spôsobom. Znova sa spoliehame na poradie obcí v csv súbore – obce sú zoradené v rovnakom poradí ako okresy, do ktorých patria. Pred samotným načítaním obcí načítame hustoty obyvateľstva každej obce (pozri 9.1 Načítanie zvolených údajov do aplikácie). Na zaradenie obce pod správny okres využívame dvojicu iterátorov. Prvý iterátor prechádza postupne kraje koreňa a druhý iterátor prechádza okresy aktuálne zvoleného kraja (kraja, na ktorý ukazujeme prvým iterátorom). Údaje spracujeme do triedy Obec a skontrolujeme či sa zhoduje kód okresu s kódom obce. Ak áno zaradíme obec do daného okresu. Ak nie posunieme iterátor okresov dopredu. V tomto momente musíme skontrolovať či sme sa nedostali na koniec prechádzaných okresov. Ak áno potom posunieme dopredu iterátor krajov a do iterátora okresov priradíme iterátor okresov aktuálneho kraja. Zložitosť takéhoto načítavania údajov je tiež O(n).



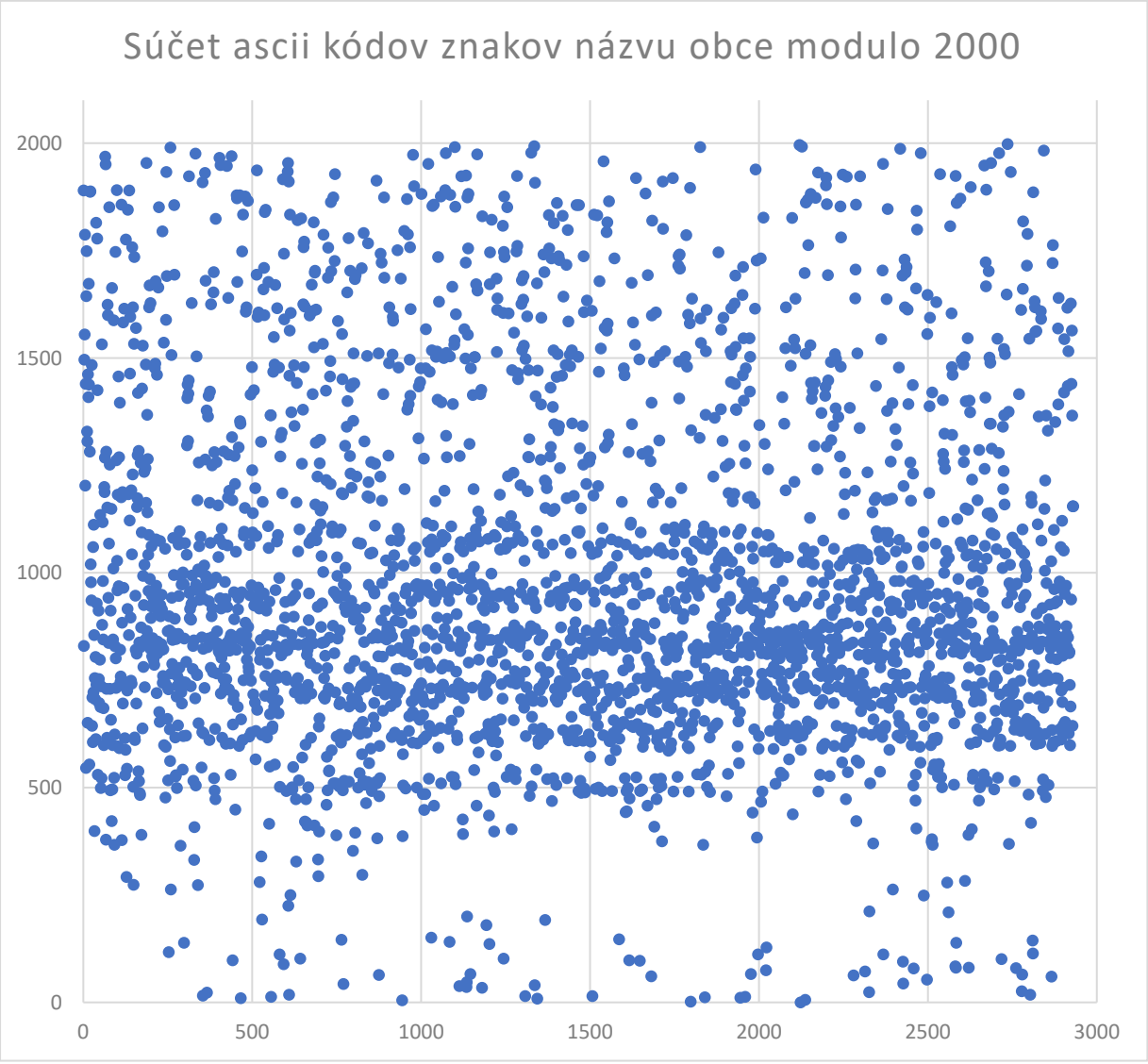
Obr. 5 Diagram aktivít algoritmu na spracovanie obcí do hierarchie

7 Implementácia tretej úrovne SP

V tretej úrovni umožňujeme používateľovi zadať názov územnej jednotky a aplikácia mu efektívne sprístupni informácie o danej územnej jednotke. Využívame pri tom pre každý typ územnej jednotky tabuľky s rozptýlenými záznamami. Používame dva typy hashovacích funkcií:

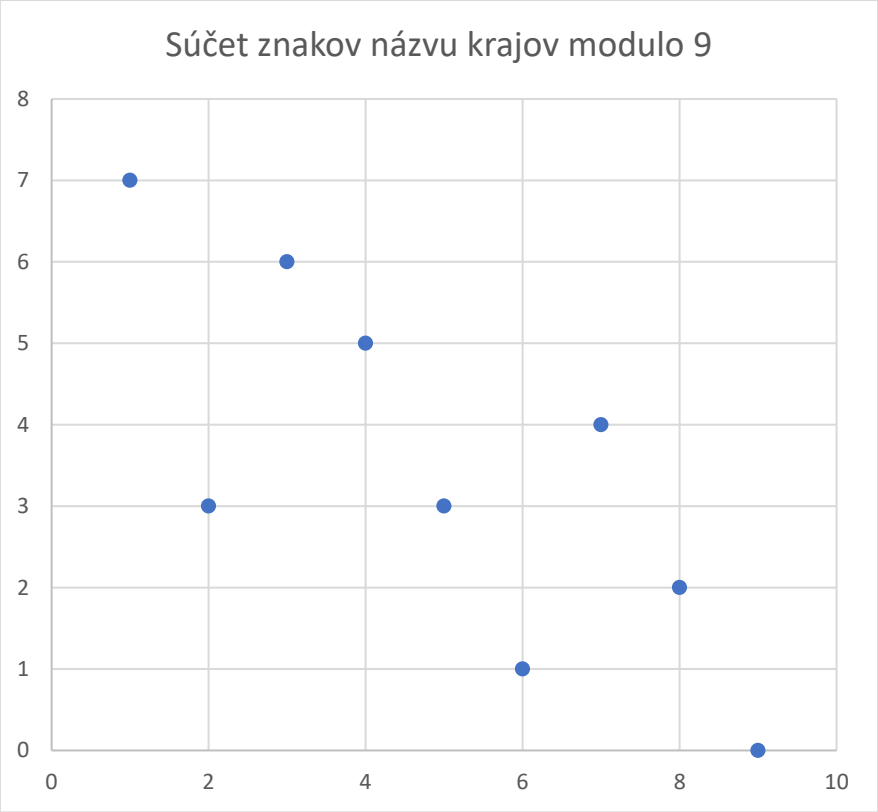
- súčet ascii kódov znakov zadaného kľúča (tabuľka obcí mod 2000 a tabuľka okresov mod 50)
- súčet znakov zadaného kľúča (tabuľka krajov mod 9)

Prvá hashovacia funkcia v prípade obcí efektívne rozptýli svoje funkčné hodnoty na intervale <0;1999> (obor hodnôt) a v prípade okresov sú jej funkčné hodnoty <0;49>. V grafe sú zobrazené funkčné hodnoty pre každú obec (os x – poradie obce, os – y funkčná hodnota hashovacej funkcie).



Obr. 6 Rozptyl hashovacej funkcie „súčet ascii kódov znakov názvu obcí modulo 2000“

V prípade tabuľky krajov, bola zvolená druhá hashovacia funkcia, ktorá každému kraju priradí unikátnu funkčnú hodnotu okrem Žilinského a Trnavského kraja (majú rovnaký počet znakov 13), tieto kraje dostanú rovnakú funkčnú hodnotu. V grafe os x prezentuje poradenie kraju (8 krajov + zahraničie) a os y sú funkčné hodnoty hashovacej funkcie.



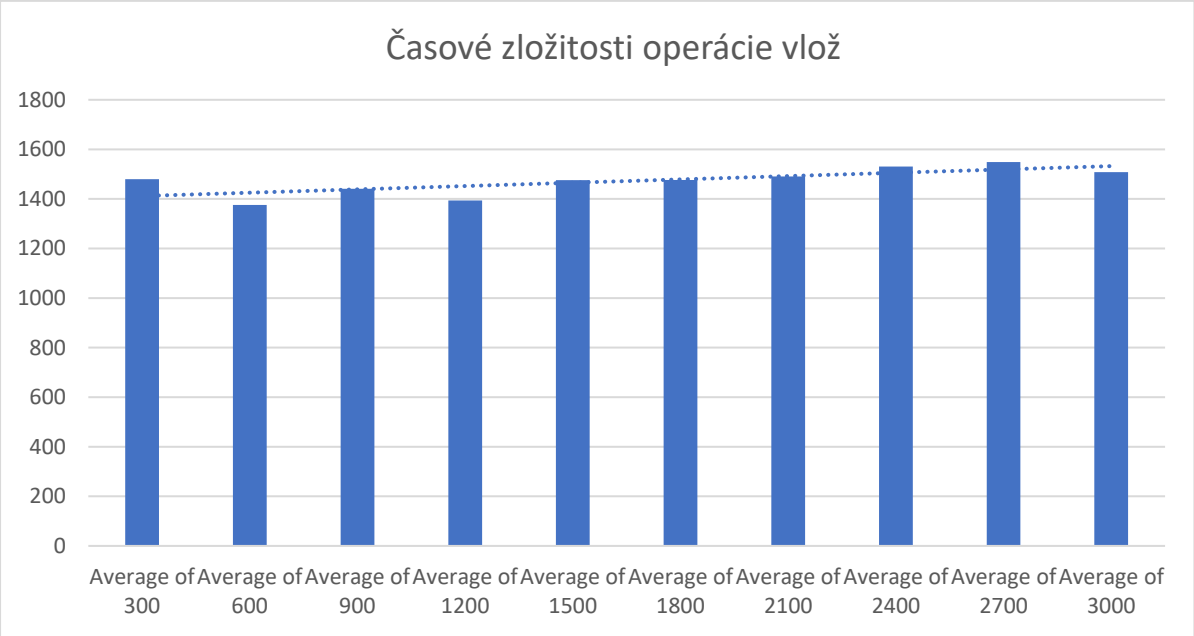
Obr. 7 Rozptyl hashovacej funkcie „súčet znakov názvu krajov modulo 9“

7.1 Riešenie duplicitných kľúčov v tabuľkách

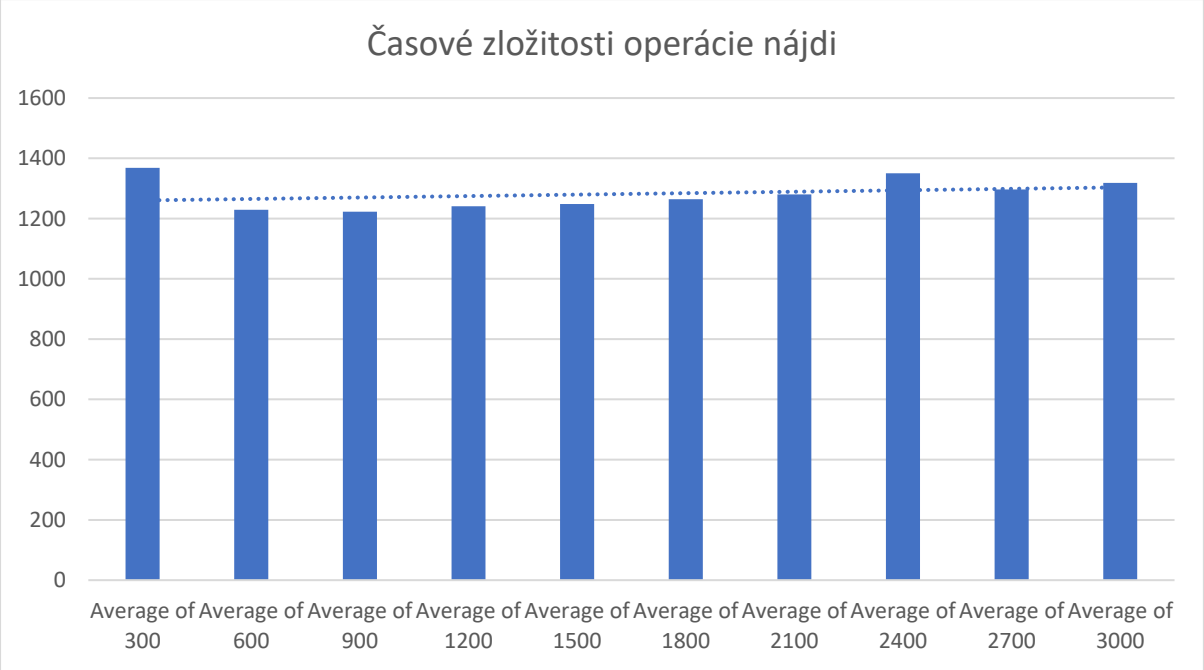
Aby sme mohli do tabuľky ukladať aj územné jednotky s rovnakým kľúčom (napr. obec Višňové), miesto danej územnej jednotky ukladáme do tabuľky implicitný list územných jednotiek s rovnakým názvom. V triede *App* používame pomocnú metódu na vkladanie prvkov do tabuľky, ktorá najskôr skontroluje, či už v tabuľke existuje prvok s daným kľúčom. Ak nie vytvorí dynamicky alokovaný implicitný list a doň vloží územnú jednotku. Ak už v tabuľke existuje položka s daným kľúčom, z tabuľky získame tento implicitný list a len doň vložíme ďalšiu územnú jednotku. Týmto spôsobom sa o niečo zvýšia pamäťové nároky ale tabuľka s použitím vyššie uvedených hashovacích funkcií poskytne rýchly prístup k prvkom tabuľky a jednoduchú prácu s duplicitnými územnými jednotkami.

7.2 Analýza časovej zložitosti tabuľky s rozptýlenými záznamami

Teoreticky by mala byť tabuľka s rozptýlenými záznamami najrýchlejšou tabuľkou. Teoretická časová zložitosť operácie vlož je O(1) a operácie nájdi tiež O(1). Pre našu hashovaciu funkciu „súčet ascii kódov znakov zadaného kľúča“ pre kľúč „názov obce“ v našom prípade potvrdzuje O(1) zložitosť v oboch meraniach.



Obr. 8 Analýza časovej zložitosti tabuľky s rozptýlenými záznamami operácie vlož



Obr. 9 Analýza časovej zložitosti tabuľky s rozptýlenými záznamami operácie nájdi

8 Implementácia štvrtej úrovne SP

Štvrtá úroveň používa vyfiltrované územné jednotky z druhej úrovne, ktoré sú uložené v implicitnej sekvencií. Za samotné triedenie je zodpovedná trieda *SortingAlgoritm*, ktorá obsahuje univerzálny triediaci algoritmus pomocou metódy *sortItemsUsing*. Do parametrov je potrebné zadať implicitnú sekvenciu a komparátor vo forme lambda funkcie alebo funkčného objektu. Z triedy *App* používame podobne ako pri filtrovacom algoritme metódu, kde stačí zadať implicitnú sekvenciu a poradové číslo komparátora. Trieda obsahuje dva komparátory na výber – usporiadanie abecedne alebo usporiadanie podľa počtu samohlások v názve územnej jednotky.

9 Bonusová úroveň

Dáta pre túto úroveň sme získali zo stránky Štatistického úradu Slovenskej republiky, konkrétne z databázy DataCube. Ako údaje boli vybrané hustoty obyvateľstva na kilometer štvorcový pre každú obec. Na stránke sa dá zvoliť aby výstup obsahoval iba kód každej obce spolu s hustotou v danom roku (bol zvolený rok 2022). Tieto údaje boli zvolené pretože sa viažu ku každej obci (okrem dvoch obcí kde údaj chýbal) a z nadradených územných celkov môžeme poskytnúť agregovaný výpis – konkrétne priemernú hustou obcí, ktoré patria pod daný nadradený územný celok.

Dáta sme exportovali ako „snímka do Excelu“ vo formáte „.xlsx“. Z Excelu sme odstránili stĺpec okresu a nepotrebné roky. Tým ostal iba stĺpec s kódmi obcí a stĺpec s hustotami obcí v roku 2022. Excel sme potom uložili ako csv súbor s kódovaním Windows-1250.

9.1 Načítanie zvolených údajov do aplikácie

Pri načítaní údajov hustoty obyvateľstva sa spoliehame na rovnaké poradie údajov v súbore csv ako je poradie obcí. Pri dvoch obciach: Valaškovce (vojenský obvod) a Javorina (vojenský obvod) údaj o hustote obyvateľstva chýba preto hustotu v týchto obciach považujeme za 0.

Najskôr načítame údaje o hustote obyvateľstva a tieto údaje uložíme do údajovej štruktúry explicitný front. Každý údaj ukladáme do inštancie triedy *PopulationDensity* s dvoma atribútmi kód obce a hustota. Potom načítame obce a vždy pri vytvorení obce a následnom zaradení obce pod správny okres a kraj porovnáme či sa kód načítanej obce zhoduje s kódom obce uloženým v triede *PopulationDensity* n vrchu explicitného frontu. Ak áno objekt zo zásobníka vyberieme a danú hustotu priradíme danej obci. Keďže aplikácia má poskytnúť agregovaný výpis nadradených územných jednotiek, pre okres a kraj zavoláme príslušnú metódu na aktualizovanie priemernej hustoty, kde pošleme danú hustotu. V tomto momente sme priradzovali okres a kraj k obci takže máme ich k dispozícií a nemusíme ich v hierarchií hľadať.

Ak by sme najskôr načítali obce a až potom hustoty, bolo by to neefektívne a vznikla by tu zložitosť $O(n^2)$. Ale našim zvoleným prístupom je zložitosť načítavania údajov O(n), čiže iba prvotné naplnenie explicitného frontu.