# Fakulta riadenia a informatiky

## Algoritmy a údajové štruktúry 1

### Dokumentácia semestrálnej práce

David Nemec

5ZYS24

2022/2023

# Návrh aplikácie

Aplikácia je zložená z 13 tried, z ktorých 3 triedy implementujú funkčnosť požadovanú úrovňami 1 až 3. Štvrtá úroveň je nadstavbou nad hierarchiou z druhej úrovne a iba pridáva utriedenie výstupu po spustení filtrovacieho algoritmu. Avšak toto triedenie bolo potrebné aj pre výstup z prvej úrovne, z tohto dôvodu je táto funkčnosť oddelená do vlastnej triedy, ktorá podľa používateľom zadaného predikátu utriedi tento výstup. Dáta o územných jednotkách (**Kraj**, **Okres** a **Obec**) sú uložené vo vlastných triedach, ktoré dedia z všeobecnej triedy **UzemnaJednotka**. Načítanie dát zabezpečuje trieda **Citac**, ktorá rozdelí vstupný súbor a uloží dáta konkrétnej územnej jednotke. Medzi ďalšie triedy patrí pomocná trieda **OblastNazov**, ktorej úlohou je postupne vrátiť názov oblasti, do ktorej patrí daný kraj a trieda **DataTabulky** v ktorej sú uložené ukazovatele na územnú jednotku (prípadne sekvencia územných jednotiek v prípade kolízie kľúča) a táto trieda je potom použitá ako hodnota daného kľúča v tabuľke. Filtrovací algoritmus je zapuzdrený vo vlastnej triede **Algorithm**. Dáta o územných jednotkách sú uložene v implicitných sekvenciách podľa typu územnej jednotky v triede **DatovaUroven**.

# Používateľská príručka

Po spustení programu je možné vybrať ktorú úroveň má program spustiť alebo je možné ho ukončiť stačením 0. Výber úrovne je zadaním čísla od 1 po 3. Štvrtá úroveň je implementovaná v 1. aj 2. úrovni, takže po zadaní parametrov pre univerzálny filtrovací algoritmus má používateľ možnosť výberu podľa akého predikátu má byť výstup zoradený.

V prvej úrovní sa program postupne spýta používateľa na hľadaný reťazec, typ územnej jednotky a predikát podľa ktorého bude výstup filtrovaný. Následne je ešte pred samotným výstupom vybrané utriedenie podľa dvoch dostupných komparátorov: utriedenie podľa abecedy alebo podľa počtu samohlások v názve. Je možné vypísať aj výstup bez utriedenia.

Druhá úroveň umožňuje navigovanie po hierarchii územných jednotiek v Slovenskej republike, kde Slovensko predstavuje koreň tejto hierarchie, jeho synovia sú oblasti, z ktorých je možné sa ďalej navigovať postupne na kraje, okresy až obce. Na začiatku je vždy vypísaná aktuálna poloha v rámci hierarchie, synovia na ktorých je možné sa presunúť z aktuálnej pozície po zadaní príkazu **son** a čísla daného syna alebo je možné sa vrátiť o úroveň vyššie zadaním **up**. Z aktuálnej pozície sa dá spustiť filtrovací algoritmus z prvej úrovne, ktorý ešte okrem dvoch predikátov (filtrovanie podľa zadaného reťazca alebo reťazca ktorým má daná územná jednotka začínať) pridáva tretí, ktorým sa dá vypísať iba územné jednotky daného typu. Filtrovanie sa spúšťa príkazom **filter**, kde druhý argument je vybraný predikát (**startsWithStr**, **containsStr**, **hasType**) a prvý je substring, podľa ktorého sú filtrované dané územné jednotky alebo typ v prípade použitia predikátu **hasType**. Výstup je ako v prípade prvej úrovne možné ešte utriediť podľa vybraného komparátora.

V tretej úrovni je najskôr zadaný kľúč podľa ktorého budú z tabuľky vypísané údaje o danej územnej jednotke a následne typ územnej jednotky. Výsledok je potom v prípade, že pre daný kľúč je iba jedna územná jednotka iba informácie o nej alebo informácia o všetkých územných jednotkách, ktoré majú daný kľúč (ako kľúč je použitý **officialTitle**).

# Rozbor vhodnosti použitia zvolených údajových štruktúr

V prvej úrovni bola ako údajová štruktúra zvolená implicitná sekvencia z dôvodu, že spĺňala podmienky požadované zadaním (zadanie požadovalo použitie sekvenčnej priamej štruktúry) a zároveň je implicitná sekvencia vhodná aj na triedenie, keďže umožňuje priamy prístup k jej prvkom pomocou indexu.

Pre uloženie krajov, okresov a obcí do hierarchie územných jednotiek Slovenskej republiky bola vybraná viaccestná hierarchia s uloženým synov v implicitnej sekvencii. Viaccestná hierarchia z dôvodu, že počet synov určitej územnej jednotky nie je vopred známy pri načítavaní (až keď sa prečítajú všetci synovia danej územnej jednotky vieme ich počet), takže nie je možné použiť hierarchiu, v ktorej už pri inicializácii musíme zadať presný počet jej synov. Taktiež nebolo možné použiť uloženie synov do explicitnej sekvencie, z dôvodu neefektívneho prístupu ku konkrétnemu synovi na základe indexu pri prechádzaní hierarchie.

Pre tretiu úroveň bola použitá tabuľka treap, ktorá využíva binárny vyhľadávací strom ako základ a zefektívňuje túto štruktúru vyvažovaním podľa priority.

# Prvá úroveň

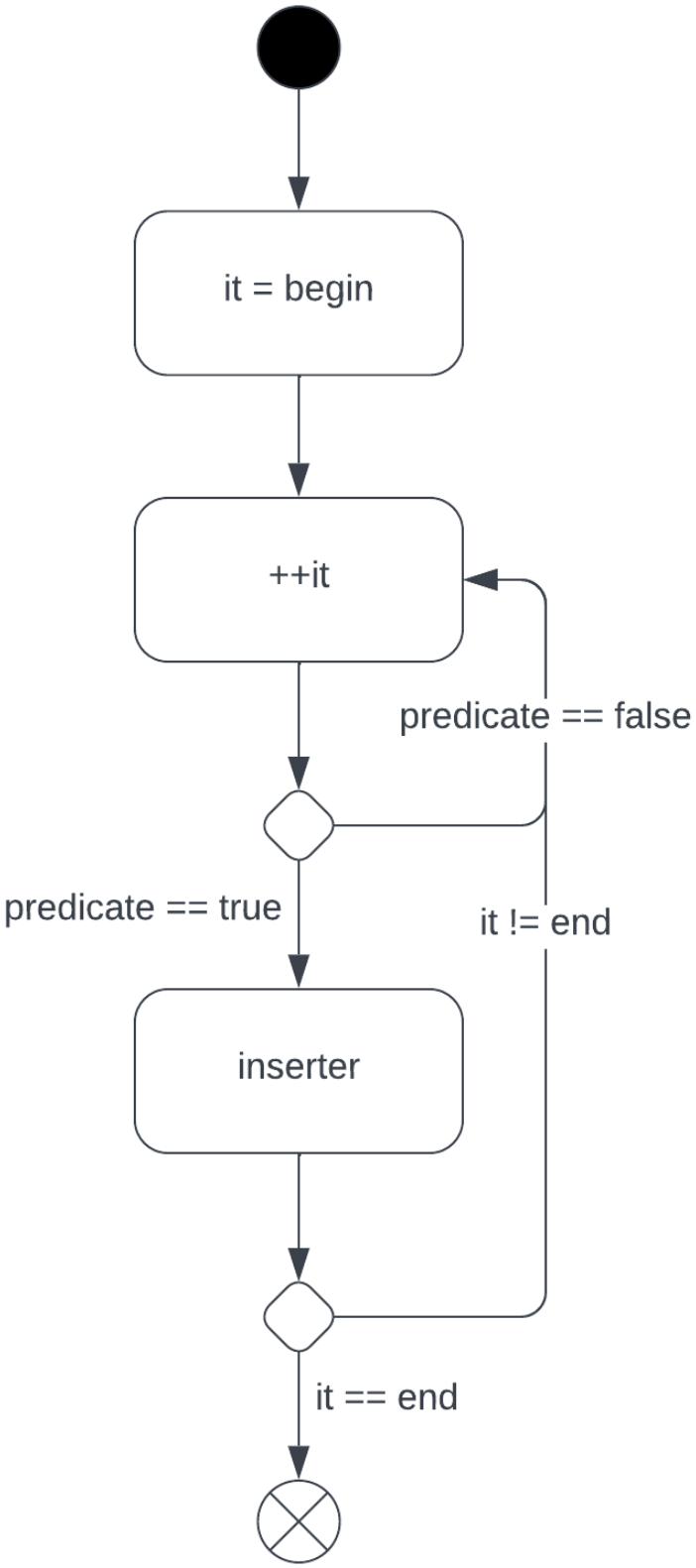
Úlohou v prvej úrovni bolo navrhnúť univerzálny filtrovací algoritmus, ktorý ako vstupné údaje má pár iterátorov a podľa zadaného predikátu vo forme lambda funkcie vyfiltruje vstupnú údajovú štruktúru a výsledok uloží do inej údajovej štruktúry, ktorá je do algoritmu vložená ako parameter.

## Programátorská príručka

Rozšíriť algoritmus o ďalší predikát je možné pridaním ďalšej premennej obsahujúcej lambda funkciu s požadovanou vlastnosťou, ak chceme v lambda funkcii sprístupniť premenné z vonkajšieho prostredia musíme použiť [&], ktorá sprístupní premenné v lambda funkcii ako referenciu. V parametroch lambda funkcie máme iba jednu premennú, ktorá ukazuje na aktuálnu územnú jednotku. Telo lambda funkcie obsahuje výraz, ktorý v prípade kladného vyhodnotenia pridá túto územnú jednotku do výslednej sekvencie. Takže môžeme v tejto lambda funkcii napríklad zadefinovať porovnanie podľa typu ako je použité v 3. úrovni pomocou dynamic_cast.

Algoritmus berie ako vstupné parametre pár iterátorov (iterátor na začiatok a iterátor na koniec), predikát vo forme lambda funkcie, referenciu na výstupnú sekvenciu (alebo aj iný údajový typ) a lambda funkciu na vkladanie do sekvencie. V tejto vkladacej funkcii môžeme podľa typu výstupnej sekvencie vložiť správnu metódu a tým pádom máme algoritmus, ktorý môže na vstupe prijať dáta v hocijakej štruktúre, ktorá má iterátory a výsledok uložiť do úplne iného dátového typu.

UML diagram aktivít



## Druhá úroveň

Do hierarchie bola doplnená z dôvodu jednoduchšieho priradzovania krajov obciam úroveň oblasť, ktorá sa nachádza medzi koreňom hierarchie a krajmi. Vkladanie do hierarchie začína pridaním koreňa – Slovensko, ktorému sú potom pridané oblasti s krajmi.

Pri načítaní krajov z implicitnej sekvencie je pre každý kraj získaný index oblasti, ktorý predstavuje 9. číslo v stĺpci **note**, napríklad pre Bratislavský kraj je **note** BL-1-SK0**10** a index oblasti je v tomto prípade **1**, keďže ale indexovanie v C++ začína od 0 je potrebné upraviť tento index aby začínal od 0. Okrem indexu oblasti, ešte **note** obsahuje aj index kraja v rámci jeho oblasti (číslo na 10. indexe, v tomto príklade je to **0**), pre ostatné oblasti ako Bratislava tento index začína od 1, takže je upravený aby zodpovedal konvencii indexovaniu v C++. Špeciálny prípad je oblasť Zahraničie, ktorá obsahuje na týchto pozíciách znak *****, takže je nutné vytvoriť podmienku, ktorá pridá tento kraj do správnej oblasti. Pri vkladaní krajov do hierarchie sú najskôr vytvárané oblasti pre dané kraje (vždy prvý kraj vytvorí oblasť kam patrí). Následne sa potom kraje vkladajú metódou **emplaceSon**, kde ako parametre sú index syna získaného z **note** a ako otec je daná oblasť, do ktorej daný kraj patrí.

Okresy sú podobne ako kraje vkladané zo sekvencie okresov načítanej na začiatku programu. Tentokrát je použitý stĺpec **code**, pomocou ktorého je možné určiť oblasť a následne kraj do ktorého daný okres patrí. Na pozícii 4 a 5 sú čísla, ktoré predstavujú indexy oblasti a kraja popísané pri vkladaní krajov, napríklad pre okres Žilina: SK0**31B**, posledné číslo predstavuje index okresu v rámci jeho kraja, toto číslo je v hexadecimálnej sústave, preto je ho potrebné najskôr previesť do desiatkovej sústavy, tento údaj sa ale nachádza aj v stĺpci **note**, kde už je v desiatkovej sústave. Z tohto dôvodu boli v súbore s okresmi vymenené riadky, aby bolo zachované poradie okresov v rámci jeho kraja (pôvodné usporiadanie radilo tie okresy, čo obsahovali čísla v hexa sústave pred tie, čo mali všetky cifry ako čísla z desiatkovej sústavy).

Načítanie obcí potom kombinuje popis v predchádzajúcich odstavcoch, najskôr cez **note** sa zistí, ktoré dáta zo stĺpca **code** predstavujú užitočnú informáciu pre určenie okresu a kraja (dôležité sú znaky na pozícii 4 až 6), napríklad obec Topoľčany s **code** SK0**236504998** a **note** **504998**, najskôr z **23** je možné určiť správny kraj (Nitriansky) a potom z čísla **6** aj okres (Topoľčany). Keďže pristupujeme stále podľa indexov a synovia v tomto type hierarchie sú uložený v implicitnej sekvencii, ktorá umožňuje efektívny prístup k jej prvkom na základe indexu, má toto vkladanie zložitosť O(1).

Celková zložitosť vkladania všetkých územných jednotiek do hierarchie je O(n), kde n je počet riadkov súboru s územnými jednotkami. Každý riadok vstupného súboru je spracovaný práve raz a je z neho vytvorený objekt danej územnej jednotky, ktorá je potom vložená do spoločnej implicitnej sekvencie prístupnej pre všetky úrovne. Vkladanie do hierarchie potom iba prechádza túto implicitnú sekvenciu a vytvára hierarchiu ukazovateľov na dané územné jednotky.

## Tretia úroveň

Typická zložitosť operácií **insert** a **find** v tabuľke **treap** je $O(\log_2(n))$, pričom najhoršia zložitosť môže byť až $O(n)$ v prípade použitia nekvalitného generátora náhodných priorít. Praktickou analýzou zložitosti bolo zistené, že pri operácií **find** odpovedá táto typická zložitosť aj reálnej implementácii (rozdiel medzi $\log_2(n)$ a nameranými dátami je minimálny). Pri vkladaní do tejto štruktúry pomocou metódy **insert** bol rozdiel oproti grafu logaritmu v niektorých miestach výrazne väčší, avšak stále nedosahovala táto metóda lineárnu zložitosť.