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

Automaticky generovaný popis

**SEMESTRÁLNA PRÁCA Z PREDMETU**

**ALGORITMY A ÚDAJOVÉ ŠTRUKTÚRY 1**

**Autor**: Šimon Lušňák

**Štúdijná skupina**: 5ZYI22

Návrh aplikácie

Aplikácia slúži na správu a filtrovanie autobusových zastávok z rôznych regionálnych dopravných systémov. Zastávky sú načítané z CSV súborov a uložené v hierarchii pre jednoduchú navigáciu a v tabuľkách pre efektívne vyhľadávanie. Aplikácia ponúka štyri úrovne interakcie pre používateľa: základné filtrovanie zastávok podľa názvu, navigáciu v hierarchii s filtrovaním podľa názvu a zoradením zastávok podľa názvu, vyhľadávanie informácii o zastávkach podľa ich názvu a zoraďovanie filtrovaných zastávok podľa názvu abecedne alebo podľa počtu spoluhlások v názve.

UML Diagram tried



Rozbor vhodnosti použitia zvolených údajových štruktúr

**HashTable**: Hash tabuľky umožňujú efektívne vyhľadávanie zastávok podľa názvu. Ich použitie je vhodné kvôli konštantnej priemernej časovej zložitosti operácií vlož a nájdi.

**MultiWayExplicitHierarchy**: Táto hierarchická štruktúra umožňuje efektívne ukladanie a navigáciu v stromovej štruktúre dopravcov, obcí a zastávok.

**ImplicitSequence**: Táto štruktúra je vhodná pre ukladanie filtrovaných výsledkov, pretože umožňuje efektívny prístup a manipuláciu s dátami.

**ImplicitList**: Pri ukladaní zastávok do HashTable s ich názvom ako kľúčom môže nastať situácia, že viacero zastávok bude mať rovnaký názov. ImplicitList v tomto prípade slúži na udržiavanie záznamov s rovnakým názvom v jednej položke v tabuľke.

**Vector**: V tomto prípade slúži vector na ukladanie zastávok načítaných zo vstupných súborov.

Používateľská príručka

**Spustenie aplikácie**: Po spustení aplikácie sa zobrazí hlavné menu s možnosťami výberu úrovne. Používateľ si následne zadaním čísla 1-4 vyberie úroveň alebo zadaním 0 ukončí program.

**Prvá úroveň**: Používateľ si najskôr zadaním čísla 1-8 zvolí dopravcu, následne môže na základe kľúča vyfiltrovať zastávky daného dopravcu, ktoré spĺňajú jeden z predikátov, ktorý si použivateľ zvolí zadaním čísla 1-2. Po zvolení predikátu používateľ zadá kľúč na základe ktorého sú zastávky vyfiltrované a následne vypísané.

**Druhá úroveň**: Umožnuje používateľovi navigovať sa v hierarchii dopravcov, obcí a zastávok zadaním 1-n pre navigáciu o úroveň nižšie alebo 0 pre úroveň vyššie. Podobne ako v prvej úrovni má možnosť filtrovania zastávok z aktuálneho vrchola zadaním „f“, po vyfiltrovaní má možnosť zoradenia výsledkov filtrovania buď podľa abecedy zadaním 1, alebo podľa počtu spoluhlások v názve zastávky zadaním 2. Ukončiť 2. úroveň môže zadaním „e“.

**Tretia úroveň**: Umožňuje vyhľadávanie informácii o konkrétnej zastávke u vybraného dopravcu podľa názvu zastávky, ktorý zadá používateľ. Používateľ najskôr zadá 1-8 pre výber dopravcu a následne názov zastávky ktorú chce vyhľadať.

**Štvrtá úroveň**: Poskytuje možnosť zoraďovania filtrovaných zastávok podľa abecedy alebo počtu spoluhlások. Táto úroveň pracuje s vyfiltrovanými zastávkami z prvej alebo druhej úrovne, pokiaľ používateľ pred spustením štvrtej úrovne teda nepoužil filtrovanie zastávok v jednej z prvých dvoch úrovní, štvrtá úroveň nemá čo zoradiť a tým pádom neumožní používateľovi použiť svoju funkcionalitu.

Zoznam použitých štruktúr implementovaných na cvičeniach

ds::adt::**HashTable**

ds::adt::**ImplicitList**

ds::amt::**MultiWayExplicitHierarchy**

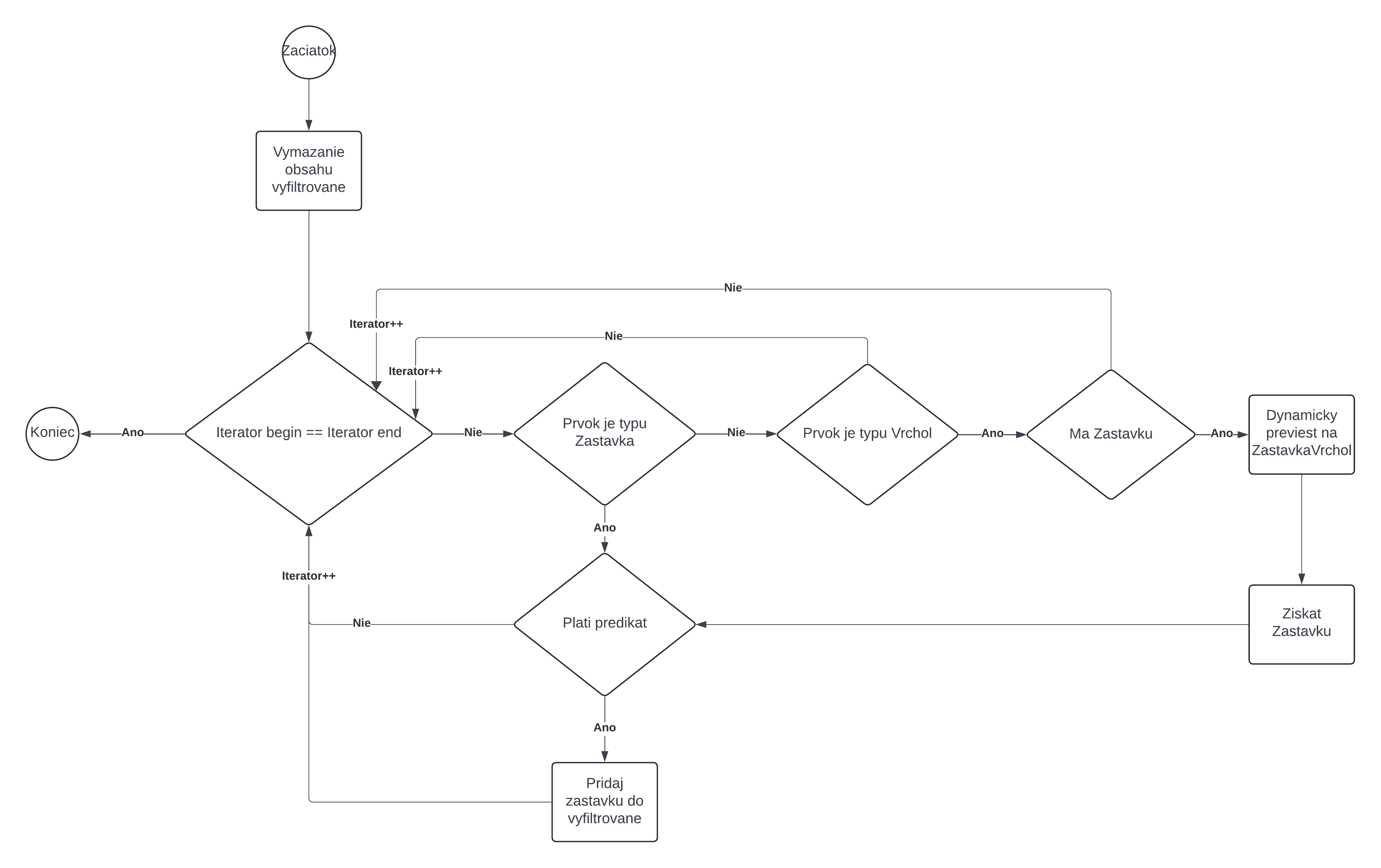
ds::amt::**MultiWayExplicitHierarchy::PreOrderHierarchyIterator**

ds::amt::**MultiWayExplicitHierarchy::PostOrderHierarchyIterator**

ds::amt::**ImplicitSequence**

ds::adt::**ShellSort**

UML diagram aktivít pre FilterZastavok



Programátorská príručka pre FilterZastavok

**Použitie univerzálneho algoritmu FilterZastavok**:

Inklúzia hlavičky: Uistite sa, že máte #include pre pre FilterZastavok.

*#include "FilterZastavok.h"*

**Volanie metódy vyfiltruj**: Algoritmus vyžaduje iterátory na začiatok a koniec kolekcie, predikát vo forme lambda funkcie, ktorá vracia bool hodnotu a kolekciu na uloženie výsledkov.

*FilterZastavok filter;*

*std::function<bool(const Zastavka&)> predikat = [](const Zastavka& z) { return /\* podmienka \*/; };*

*filter.vyfiltruj(zastavky.begin(), zastavky.end(), predikat, vyfiltrovane);*

**Rozšírenie algoritmu o nový predikát**:

Definícia nového predikátu: Môžete definovať nový predikát ako lambda funkciu, ktorá vracia bool hodnotu

auto novyPredikat = [](const Zastavka& z) { return /\* nová podmienka \*/; };

**Použitie nového predikátu**: Nový predikát môžete použiť priamo pri volaní vyfiltruj.

*filter.vyfiltruj(zastavky.begin(), zastavky.end(),* ***novyPredikat****, vyfiltrovane);*

Úpravy vstupných súborov

Kvôli zachovaniu zložitosti načítavania údajov zo vstupných súborov som upravil vstupné súbory tak,

že som zoradil záznamy abecedne podľa stĺpca „*muni*“ teda podľa názvu Obce.

Ďalej v záznamoch, v ktorých sa pred a za názvom zastávky nachádzali úvodzovky som úvodzovky odstránil. Podobnú úpravu som urobil aj pri názvoch zastávok ktoré pred názvom obsahovali medzeru.

Dôvodom bolo, že by mohlo dôjsť k problémom pri vyhľadávaní zastávok podľa názvu v tabuľkách v 3. úrovni.

Spôsob a zložitosť načítavania údajov

Trieda **DataLoader** a jej metóda **nacitajData** slúžia na načítanie údajov zo CSV súborov a ich uloženie do príslušných štruktúr. Načítavanie údajov prebieha v týchto krokoch:

**Otváranie súboru**: Súbor je otvorený pomocou std::ifstream.

**Čítanie riadkov**: Každý riadok súboru je čítaný a rozdelený na jednotlivé hodnoty.

**Vytvorenie objektu Zastavka**: Na základe hodnôt z riadku je vytvorený objekt Zastavka.

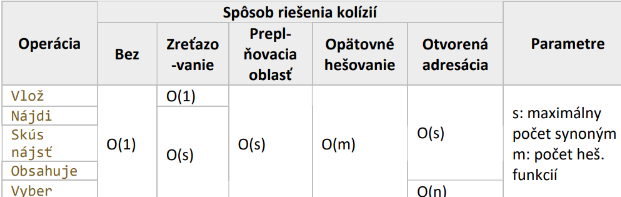
**Uloženie do std::vector a tabuľky**: Objekt je uložený do vektora a tabuľky príslušného dopravcu.

**Vytvorenie vrcholov**: Na začiatku sa vytvorí vrchol pre daného dopravcu a postupne sa mu pridávajú vrcholy pre obce, ktorým sa následne pridávajú vrcholy pre ich prislúchajúce zastávky. Keďže sú záznamy vo vstupných súboroch zoradené abecedne podľa názvu obce, nový vrchol pre obec vytvoríme iba v prípade, že sa jej názov líši od tej predošlej.

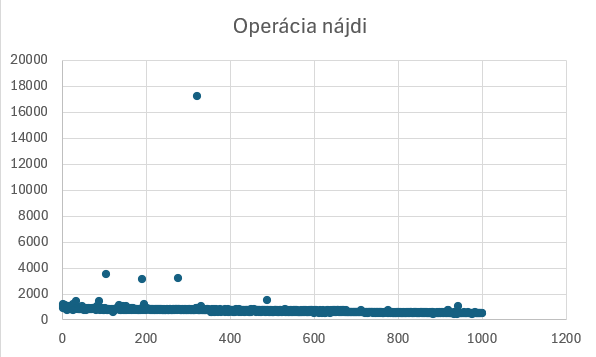
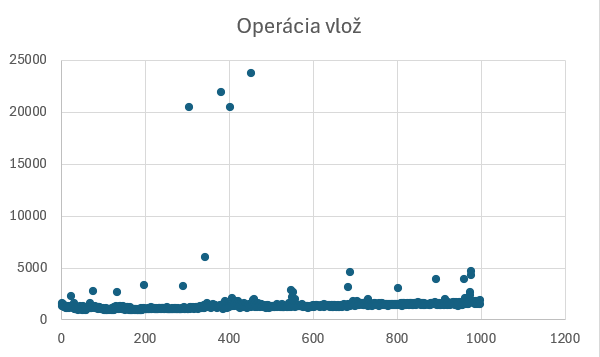
**Zložitosť načítavania**: Časová zložitosť**:** **O(n)**, kde n je počet riadkov v súbore.

Analýza časovej zložitosti operácie vlož a nájdi pre HashTable

Teoretická zložitosť:



Empirická zložitosť:



Zvislá os v grafoch predstavuje čas v nanosekundách a vodorovná os predstavuje počet meraní.

Z grafov môžeme usúdiť, že pri oboch operáciach je časová zložitosť konštantná.