

Semestrálna práca z predmetu *algoritmy a údajové štruktúry 1*

Iar

**vypracovali:** Ester Holeščáková

**študijná skupina:** 5ZYR21

**cvičiaci:** Ing. Michal Mrena PhD.

**termín cvičenie:** utorok bloky 4-6 v Žiline dňa 01.05.2025

1. Návrh aplikácie

Aplikácia je navrhnutá tak, aby jednoducho a prehľadne spravovala informácie o autobusových zastávkach. Funguje ako digitálny asistent, ktorý pomáha rýchlo nájsť, usporiadať a prezerať dáta o zastávkach.

Hlavným cieľom bolo zabezpečiť flexibilitu a rýchlosť aplikácie. To znamená, že aj keď sa zmenia požiadavky na vyhľadávanie alebo triedenie dát, aplikácia sa ľahko prispôsobí bez rozsiahlych zmien v základnom kóde.

**Architektúra Aplikácie**

Aplikácia je štruktúrovaná do niekoľkých logických komponentov, ktoré spolupracujú na spracovaní dát:

1. **Načítanie Dát:**
   * Informácie o zastávkach (ID, adresa, geografická poloha) sa načítavajú z určeného súboru. Tieto dáta sú následne uložené do hlavného zoznamu, ktorý slúži ako primárny zdroj pre všetky ďalšie operácie.
2. **Filtrovanie Dát:**
   * Implementovaný je univerzálny mechanizmus na vyhľadávanie zastávok podľa definovaných kritérií. Možno filtrovať podľa obce, ulice alebo geografickej oblasti.
   * Tento filtračný algoritmus je vysoko prispôsobivý vďaka použitiu predikátov, ktoré možno dynamicky meniť. Takto je možné realizovať rôzne typy vyhľadávania pomocou toho istého nástroja. Výsledkom je vždy nový zoznam nájdených zastávok.
3. **Hierarchické Usporiadanie Dát:**
   * Pre zlepšenie orientácie v dátach sú zastávky usporiadané do stromovej štruktúry. Koreň reprezentuje dopravcu (**G**rand **R**iver **T**ransit), nasledujú uzly pre obce a pod nimi uzly pre ulice. Konkrétne zastávky sú priradené k uzlom ulíc.
   * Táto hierarchia umožňuje navigáciu dátami (pohyb medzi obcami a ulicami).
   * Funkcionalita filtrovania je dostupná aj v rámci ľubovoľného podstromu hierarchie, čo umožňuje cielené vyhľadávanie v špecifických geografických oblastiach.
4. **Rýchle Vyhľadávanie podľa ID:**
   * Pre efektívny prístup k informáciám o konkrétnej zastávke podľa jej ID je implementovaná hašovacia tabuľka.
   * Táto dátová štruktúra zabezpečuje takmer okamžité vyhľadanie zastávky na základe jej identifikátora, čím sa eliminuje potreba sekvenčného prehľadávania celého zoznamu dát.
5. **Univerzálne Triedenie Dát:**
   * Vyfiltrované zoznamy zastávok možno zoradiť. K dispozícii sú kritériá triedenia podľa abecedného poradia obcí (a poprípade ulíc), alebo podľa číselného ID zastávok.
   * Triediaci algoritmus je navrhnutý s vysokou flexibilitou, umožňujúcou dynamické nastavenie komparátora. Využíva efektívny triediaci algoritmus QuickSort, čo zaručuje rýchle spracovanie aj rozsiahlych dát.

**Tok dát v aplikácií**

Hlavný tok dát v aplikácii začína načítaním zo súboru a prechádza cez rôzne úrovne spracovania a reprezentácie:

**1. Načítanie dát**

* **Zdroj:** GRT\_Stops.csv (CSV súbor s dátami o zastávkach)
* **Štruktúra:** std::vector<StopData> allStops
* **Proces:** Trieda FileReader načíta riadky zo súboru. Každý riadok parsuje a konvertuje do objektu StopData. Tieto objekty sú následne uložené do std::vector<StopData>. Tento vektor slúži ako primárny zdroj všetkých dát o zastávkach pre celú aplikáciu.

**2. Inicializácia štruktúr a filtrovanie**

* **Zdroj:** std::vector<StopData> allStops
* **Štruktúry:** std::vector<StopData>
* **Proces:**
  + ConsoleManager dostane referenciu na allStops.
  + Pri požiadavke na filtrovanie, StopFilter dostane iterátory k allStops.
  + StopFilter prechádza cez dáta vo allStops a na základe zvoleného predikátu (lambda funkcia) vyberá zodpovedajúce StopData objekty.
  + Vyfiltrované zastávky sú uložené do nového std::vector<StopData>, ktorý je potom vypísaný.

**3. Budovanie hierarchie a navigácia**

* **Zdroj:** std::vector<StopData> allStops
* **Štruktúra:** ds::adt::MultiwayTree<NodeData>
* **Proces:**
  + HierarchyBuilder dostane allStops. Vytvorí koreň stromu pre dopravcu ("GRT").
  + Pre každú StopData z allStops, HierarchyBuilder buď nájde existujúci uzol pre obec a ulicu, alebo ich vytvorí ako nových synov v strome.
  + Každý uzol reprezentujúci ulicu (NodeData) obsahuje ds::adt::SinglyLinkedList<StopData>\* stops, kde sú uložené kópie konkrétnych zastávok na danej ulici.
  + HierarchyNavigator umožňuje pohybovať sa po tomto strome.
  + Pri filtrovaní v hierarchii, HierarchyNavigator rekurzívne zbiera všetky StopData objekty z podstromu (z ich SinglyLinkedList v listoch) do dočasného std::vector<StopData>. Tento vektor je potom odovzdaný na StopFilter pre ďalšie filtrovanie.

**4. Vyhľadávanie podľa ID**

* **Zdroj:** std::vector<StopData> allStops
* **Štruktúra:** StopHashTable
* **Proces:**
  + Pri spustení aplikácie, StopHashTable je populovaná s dátami z allStops. Pre každú zastávku sa jej id stane kľúčom a ukazovateľ na StopData sa uloží ako hodnota.
  + Keď používateľ zadá ID, StopHashTable rýchlo vyhľadá zodpovedajúci ukazovateľ na StopData pomocou hašovacej funkcie.

**5. Triedenie dát**

* **Zdroj**: std::vector<StopData>
* **Štruktúra:** ds::amt::ImplicitSequence<StopData>
* **Proces:**
  + ConsoleManager odovzdá vyfiltrovaný std::vector<StopData> triede StopSorter.
  + StopSorter skopíruje dáta z std::vector<StopData> do dočasnej štruktúry ds::amt::ImplicitSequence<StopData>.
  + Na tejto ImplicitSequence je zavolaný triediaci algoritmus (ds::adt::QuickSort), ktorý používa lambda funkciu ako komparátor.
  + Zoradené dáta zostávajú v ImplicitSequence a sú z nej potom vypísané.

1. Obrázok, na ktorom je text, nalepovacie lístočky, snímka obrazovky, papierový výrobok

   Obsah vygenerovaný umelou inteligenciou môže byť nesprávny.UML diagram tried
2. Rozbor vhodnosti použitia zvolených údajových štruktúr

**Zadanie 1: Načítanie a filtrovanie dát**

* **std::vector<StopData>**:
  + **Použitie**: Slúži ako primárna štruktúra na počiatočné načítanie všetkých dát o zastávkach zo súboru. Následne sa používa aj na uloženie výsledkov operácií filtrovania.
  + **Vhodnosť**: Je to sekvenčná dátová štruktúra s priamym prístupom, ktorá je veľmi efektívna pre iteráciu a sekvenčné spracovanie. Pre načítanie dát je std::vector optimálny vďaka efektívnemu pridávaniu prvkov na koniec (O(1)). Pre filtrovanie je výhodný, keďže poskytuje požadovanú sekvenčnú štruktúru pre vstup aj výstup univerzálneho filtračného algoritmu.

**Zadanie 2: Hierarchické usporiadanie dát**

* **ds::adt::MultiwayTree<NodeData>**:
  + **Použitie**: Táto dátová štruktúra predstavuje hierarchiu zastávok – od koreňa (dopravca GRT) cez obce až po ulice.
  + **Vhodnosť**: Viaccestný strom je ideálna voľba pre reprezentáciu hierarchických dát. Umožňuje logické zoskupovanie zastávok a efektívnu navigáciu medzi úrovňami (obce, ulice). Poskytuje priamy prístup k deťom uzla, čo je kľúčové pre implementáciu navigátora hierarchie.
* **ds::adt::SinglyLinkedList<StopData> v NodeData (v uzloch ulíc)**:
  + **Použitie**: Na najnižšej úrovni hierarchie (v uzloch reprezentujúcich ulice) sú konkrétne zastávky uložené v týchto zoznamoch.
  + **Vhodnosť**: Výber jednosmerného zoznamu je vhodný pre efektívne vkladanie nových zastávok na koniec (O(1)) počas budovania hierarchie. Je tiež jednoduchý na iteráciu pri zbere všetkých zastávok z podstromu pre následné filtrovanie.

**Zadanie 3: Efektívne vyhľadávanie podľa ID**

* **ds::adt::HashTable<int, StopData\*>**:
  + **Použitie**: Táto štruktúra je kľúčová pre efektívnevyhľadávanie konkrétnej zastávky na základe jej ID.
  + **Vhodnosť**: Hašovacia tabuľka je optimálnavoľba pre operácie vyhľadávania, vkladania a mazania, s priemernou časovou zložitosťou O(1). Ukladanie ukazovateľov namiesto celých kópií objektov minimalizuje pamäťovú réžiu a zabezpečuje priamy prístup k pôvodným dátam zastávok.

**Zadanie 4: Triedenie Dát**

* **ds::amt::ImplicitSequence<StopData>**:
  + **Použitie**: Táto štruktúra je využitá ako **dočasný kontajner** pre dáta, ktoré sa majú triediť. Dáta z vyfiltrovaného std::vector (výstup z úrovne 1 alebo 2) sa do nej skopírujú pred samotným triedením.
  + **Vhodnosť**: Implicitná sekvencia je dátová štruktúra založená na poli, čo je ideálne pre triediace algoritmy pracujúce na mieste (in-place) a pre poskytovanie efektívneho priameho prístupu k prvkom.
* **ds::adt::Sort<StopData> (konkrétne ds::adt::QuickSort<StopData>)**:
  + **Použitie**: Reprezentuje triediaci algoritmus, ktorý je aplikovaný na dáta v ImplicitSequence.
  + **Vhodnosť**: QuickSort je štandardný a veľmi efektívny triediaci algoritmus s priemernou časovou zložitosťou O(n log n) - zaručuje rýchle triedenie aj pre veľké množstvo dát.

1. Používateľská príručka

**1. Spustenie aplikácie**

Po spustení aplikácia automaticky načíta dáta o zastávkach zo súboru. Môžete očakávať nasledujúce výpisy:

* **Úspešné načítanie:**

Loading stops data...

Successfully loaded XXXX stops.

(Tu XXXX reprezentuje celkový počet úspešne načítaných zastávok.)

* **Chyba pri otváraní súboru:** Ak sa súbor GRT\_Stops.csv nenájde alebo ho nie je možné otvoriť.

Loading stops data...

Failed to open file: GRT\_Stops.csv

Fatal Error: Failed to open file: GRT\_Stops.csv

No stops loaded. Exiting.

(Aplikácia sa ukončí.)

* **Prázdny CSV súbor:** Ak sa súbor nájde, ale neobsahuje žiadne dáta (alebo len hlavičku bez riadkov s dátami).

Loading stops data...

CSV file is empty: GRT\_Stops.csv

Successfully loaded 0 stops.

No stops loaded. Exiting.

* **Chyby v riadkoch CSV súboru:** Ak sú v súbore nekompletné alebo chybné riadky (napr. chýbajúce ID, ulica, obec, alebo nesprávny formát čísla). Chybný riadok sa preskočí, ale aplikácia pokračuje v načítavaní ostatných riadkov.

(Chybné riadky sa do počtu úspešne načítaných zastávok nezapočítavajú.)

Loading stops data...

Error processing line 123: 1;0;...;missing;;... - Missing StopId

Successfully loaded XXXX stops.

**2. Úroveň 1: Filtrovanie dát**

Táto sekcia umožňuje filtrovať zastávky z celého pôvodného zoznamu podľa rôznych kritérií.

**Postup**:

* Po úvodnom načítaní budete vyzvaní vybrať typ filtra: Filter by [m]unicipality, [s]treet, [r]egion, or [q]uit:
  + Zadajte m (alebo M) pre obec (municipality).
  + Zadajte s (alebo S) pre ulicu (street).
  + Zadajte r (alebo R) pre región (region).
  + Zadajte q (alebo Q) pre ukončenie.
  + Ak zadáte iné písmeno, filter sa preskočí a zobrazí sa hlásenie: Unknown filter type. Skipping this filter.
* Zadajte požadované kritériá na základe zvoleného typu filtra:
  + Pre obec (m): Enter municipality: (Zadajte presný názov obce, napr. Waterloo. Porovnanie nerozlišuje veľkosť písmen.)
  + Pre ulicu (s): Enter street (substring): (Zadajte podreťazec názvu ulice, napr. King. Porovnanie nerozlišuje veľkosť písmen.)
  + Pre región (r): Budete vyzvaní zadať 4 číselné hodnoty:
    - Enter min latitude: (Minimálna zemepisná šírka)
    - Enter max latitude: (Maximálna zemepisná šírka)
    - Enter min longitude: (Minimálna zemepisná dĺžka)
    - Enter max longitude: (Maximálna zemepisná dĺžka)
  + Ak zadáte neplatnú číselnú hodnotu, budete vyzvaní zadať ju znova: Invalid input. Please enter a number:
* Po vykonaní filtra sa zobrazí zoznam nájdených zastávok. Ak sa nenájdu žiadne zastávky, zobrazí sa No stops to display..
* Po zobrazení výsledkov budete mať možnosť zoradiť ich (viď Úroveň 4).
* Príklad filtrovania:

=========== Level 1: FILTERING ===========

Filter by [m]unicipality, [s]treet, [r]egion, or [q]uit: s

Enter street (substring): king

=========== Stops matching street containing 'king' ===========

ID: 2516, LAT: 43.4687, LON: -80.5283, STREET: King, MUN: Waterloo

ID: 2470, LAT: 43.4688, LON: -80.5428, STREET: King, MUN: Waterloo

ID: 2883, LAT: 43.4667, LON: -80.5204, STREET: King, MUN: Kitchener

... (ďalšie zastávky)

---------------------------------

Total stops displayed: XXX

**3. Úroveň 2: Navigácia v hierarchii zastávok**

V tejto úrovni môžete prechádzať hierarchickou štruktúrou zastávok (GRT 🡪 Obec 🡪 Ulica) a filtrovať dáta v rámci aktuálneho podstromu.

* **Dostupné príkazy:**

Každá interakcia začína zobrazením informácií o aktuálnom uzle a výzvou na zadanie príkazu:

Current node: <Názov uzla>

(<Počet zastávok v podstrome> stops, <Počet detí> children)

Commands: [c]hild, [p]arent, [l]ist, [f]ilter, [q]uit >

* l (list): Vypíše zoznam všetkých podradených uzlov (detí) aktuálneho uzla s ich číselnými indexmi.

Commands: [c]hild, [p]arent, [l]ist, [f]ilter, [q]uit > l

[0] Cambridge

[1] Kitchener

[2] Waterloo

[3] ... (ďalšie obce)

* c (child): Presunie sa na zvolený podradený uzol.
  + Po zadaní c budete vyzvaní zadať index dieťaťa: Enter child index:
  + Ak zadáte neplatný index (napr. mimo rozsahu), zobrazí sa hlásenie: Invalid index.

Commands: [c]hild, [p]arent, [l]ist, [f]ilter, [q]uit > c

Enter child index: 2

Current node: Waterloo

(XXX stops, X children)

* p (parent): Vráti sa na nadradený uzol - rodič.
  + Ak ste už v koreňovom uzle ("GRT"), zobrazí sa hlásenie: Already at root.

Commands: [c]hild, [p]arent, [l]ist, [f]ilter, [q]uit > p

Current node: GRT

(XXXX stops, X children)

* f (filter): Spustí filtrovanie zastávok, ale len v rámci celého podstromu, ktorého koreňom je aktuálny uzol.
  + Postup filtrovania (výber typu filtra m/s/r a zadávanie kritérií) je identický s Úrovňou 1.
  + Po filtri sa zobrazia nájdené zastávky z daného podstromu a následne môžete výsledok zoradiť (viď Úroveň 4).

Commands: [c]hild, [p]arent, [l]ist, [f]ilter, [q]uit > f

Filter by [m]unicipality, [s]treet, [r]egion: s

Enter street (substring): king

Filtered stops from subtree matching street containing 'king'

ID: 2516, LAT: 43.4687, LON: -80.5283, STREET: King, MUN: Waterloo

... (ďalšie zastávky z podstromu)

---------------------------------

Total stops displayed: XX

* q (quit): Ukončí navigáciu v hierarchii a prejde na Úroveň 3.

Commands: [c]hild, [p]arent, [l]ist, [f]ilter, [q]uit > q

Exiting Level 2...

**4. Úroveň 3: Vyhľadávanie zastávok podľa ID**

Táto úroveň umožňuje rýchlo nájsť konkrétnu zastávku pomocou jej jedinečného ID, s využitím hašovacej tabuľky.

**Postup:**

* Budete vyzvaní zadať ID: Enter a stop ID to search (or type 'q' to exit Level 3):
* Zadajte číselné ID zastávky, ktorú hľadáte.
* Výsledky:
  + Zastávka nájdená: Aplikácia zobrazí kompletné detaily zastávky.

Enter a stop ID to search (or type 'q' to exit Level 3): 2516

Stop found: ID: 2516, LAT: 43.4687, LON: -80.5283, STREET: King, MUN: Waterloo

* + Zastávka nenájdená: Zobrazí hlásenie, že zastávka s daným ID nebola nájdená.

Enter a stop ID to search (or type 'q' to exit Level 3): 10

Stop with ID 10 not found.

* + Neplatný vstup: Ak zadáte niečo, čo nie je číslo (okrem q), zobrazí sa chybové hlásenie.

Enter a stop ID to search (or type 'q' to exit Level 3): a

Invalid input. Please enter a valid stop ID or 'q'.

* Pre opustenie úrovne zadajte q.

**5. Úroveň 4: Triedenie dát**

Táto úroveň sa ponúka po každom úspešnom filtrovaní zastávok (či už v Úrovni 1 alebo v Úrovni 2), ak zoznam filtrovaných zastávok nie je prázdny.

**Postup:**

* Aplikácia sa vás opýta, či chcete zoradiť filtrované zastávky: Do you want to sort these stops? ([y]es/[n]o):
  + Zadajte y (áno) pre triedenie.
  + Zadajte n (nie) pre preskočenie triedenia.
  + Ak je zoznam filtrovaných zastávok prázdny, zobrazí sa No stops to sort. a triedenie sa preskočí.
* Ak zvolíte y, budete vyzvaní vybrať kritérium triedenia:

Choose sorting criteria:

[m] - Sort by Municipality (Street)

[i] - Sort by ID

Enter choice:

* + Zadajte m pre triedenie najprv podľa obce (abecedne) a potom podľa ulice (ak sú obce rovnaké). Zobrazí sa správa: Sorting by Municipality (street)...
  + Zadajte i pre triedenie podľa ID (číselne, vzostupne). Zobrazí sa správa: Sorting by ID...
  + Ak zadáte iné písmeno, zobrazí sa hlásenie: Invalid sorting criteria. Please enter 'm' or 'i'. No sorting applied. a triedenie sa nevykoná.
* Po úspešnom triedení sa zobrazí zoradený zoznam zastávok pod hlavičkou Sorted Stops.
* Príklad triedenia:

Do you want to sort these stops? ([y]es/[n]o): y

=========== Sorting Stops ===========

Choose sorting criteria:

[m] - Sort by Municipality (Street)

[i] - Sort by ID

Enter choice: m

Sorting by Municipality (street)...

Sorted Stops

ID: 2470, LAT: 43.4688, LON: -80.5283, STREET: King, MUN: Waterloo

ID: 2516, LAT: 43.4687, LON: -80.5283, STREET: King, MUN: Waterloo

... (ďalšie zoradené zastávky)

---------------------------------

Total stops displayed: XXX

1. Prvá úroveň

**UML DIAGRAM AKTIVÍT UNIVERZÁLNEHO ALGORITMU**

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

Obsah vygenerovaný umelou inteligenciou môže byť nesprávny.

**PROGRAMÁTORSKÁ PRÍRUČKA**

* **ako používať univerzálne navrhnutý algoritmus: *StopFilter::filter***

Univerzálny filtračný algoritmus je navrhnutý tak, aby fungoval s akýmkoľvek typom sekvenčnej dátovej štruktúry, ktorá poskytuje štandardné iterátory (napríklad std::vector::begin() a std::vector::end()). Kľúčovým prvkom jeho flexibility je predikát, teda funkcia, ktorá definuje konkrétne filtračné kritérium.



Iterator begin, Iterator end: Tieto parametre určujú rozsah dát, ktoré sa majú filtrovať. Mali by to byť iterátory na začiatok a koniec kolekcie, ktorá obsahuje objekty typu StopData.

Predicate pred: Toto je funkčný objekt (najčastejšie lambda funkcia alebo objekt typu std::function), ktorý prijíma konštantnú referenciu na objekt StopData a vracia logickú hodnotu bool.

* Ak predikát vráti true, zastávka spĺňa filtračné kritérium a bude zahrnutá do výsledného zoznamu.
* Ak predikát vráti false, zastávka nespĺňa kritérium a bude vynechaná.

V aplikácii sa StopFilter::filter volá s predikátmi, ktoré sú vytvorené statickými metódami triedy StopFilter (isInMunicipality, isOnStreet, isInRegion)

Obrázok, na ktorom je text, snímka obrazovky, písmo

Obsah vygenerovaný umelou inteligenciou môže byť nesprávny.

* **ako postupovať v prípade jeho budúceho rozšírenia o iný predikát:**

Rozšírenie filtračného algoritmu o nové kritérium je veľmi jednoduché a nevyžaduje žiadne zmeny v samotnej metóde StopFilter::filter. Stačí pridať novú statickú metódu do triedy StopFilter, ktorá vráti funkčný objekt (std::function<bool(const StopData&)>) s požadovanou logikou.

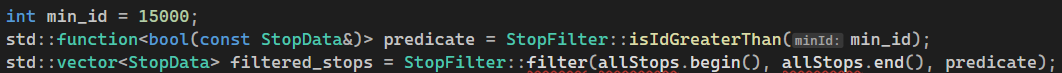
**Postup** pre pridanie nového predikátu (napr. filtra pre zastávky s ID väčším ako daná hodnota):

* V rámci triedy StopFilter pridajte novú statickú metódu. Táto metóda bude prijímať potrebné parametre pre definovanie kritéria a vracať lambda funkciu.

Obrázok, na ktorom je text, snímka obrazovky, písmo

Obsah vygenerovaný umelou inteligenciou môže byť nesprávny.

* Teraz môžete tento nový predikát odovzdať metóde StopFilter::filter rovnako ako ktorýkoľvek iný.



1. Druhá úroveň

**VYSVETLENIE SPÔSOBU NAČÍTAVANIA ÚDAJOV:**

Budovanie hierarchie je riadené triedou HierarchyBuilder, ktorá prijíma kompletný zoznam všetkých načítaných zastávok (std::vector<StopData> allStops) ako svoj vstup. Cieľom je vytvoriť viaccestný strom (ds::adt::MultiwayTree<NodeData>), ktorý logicky zoskupí zastávky.

**Postup budovania hierarchie:**

* Inicializácia koreňa stromu:
  + Proces začína vytvorením koreňového uzla stromu. Tento uzol je pomyselným reprezentantom dopravcu a je pomenovaný "GRT". Je to najvyššia úroveň hierarchie.
* Iterácia cez všetky zastávky:
  + HierarchyBuilder prechádza sekvenčne (riadok po riadku) cez každú jednotlivú zastávku (StopData) z pôvodného zoznamu allStops. Každá zastávka je spracovaná len raz.
* Vkladanie obcí:
  + Pre každú spracovávanú zastávku sa najprv skontroluje, či uzol reprezentujúci jej obec (stop.municipality) už existuje ako dieťa koreňového uzla "GRT".
  + Ak uzol pre danú obec neexistuje, vytvorí sa nový uzol (NodeData) s názvom obce a pridá sa ako dieťa koreňovému uzlu.
  + Ak uzol pre obec už existuje, jednoducho sa naň prejde.
* Vkladanie ulíc:
  + Po nájdení alebo vytvorení uzla pre obec sa rovnaký proces zopakuje pre ulicu (stop.street). Skontroluje sa, či uzol reprezentujúci danú ulicu už existuje ako dieťa aktuálneho uzla obce.
    - Ak uzol pre danú ulicu neexistuje, vytvorí sa nový uzol (NodeData) s názvom ulice a pridá sa ako dieťa uzlu obce.
    - Ak uzol pre ulicu už existuje, prejde sa naň.
* Priradenie zastávok k uliciam:
  + Zastávky nie sú samotné uzly v strome. Namiesto toho, každý uzol, ktorý reprezentuje ulicu (listové uzly stromu), obsahuje pointer na ds::adt::SinglyLinkedList<StopData>.
  + Keď sa spracuje zastávka a nájde sa jej príslušný uzol ulice, kópia objektu StopData tejto zastávky sa pridá do SinglyLinkedList tohto uzla ulice. Týmto spôsobom sú všetky zastávky prislúchajúce danej ulici zoskupené v rámci jedného listového uzla stromu.

**ZLOŽITOSŤ NAČÍTAVANIA:**

* **Iterácia cez zastávky:**
  + Hlavná slučka v HierarchyBuilder::build prechádza cez každú z n zastávok presne raz aby bola spracovaná - **O(n)**.
* **Vyhľadávanie/vkladanie uzlov:**
  + Pre každú zastávku sa musí vykonať operácia nájdenia alebo vytvorenia uzla pre obec a uzla pre ulicu.
  + Vkladanie nového uzla na koniec zoznamu detí je **O(1)**.
  + Hľadanie existujúceho uzla medzi deťmi rodiča môže v najhoršom prípade trvať toľko operácií, koľko má rodič detí.
  + Zo zadania: „Môžete sa spoľahnúť na poradie údajov vo vstupnom CSV súbore“. (Zastávky sú naozaj v súbore zoradené abecedne podľa street – ulice, teda aj vo vektore sú tak uložené). Potom je pravdepodobnosť, že uzol už bude existovať, veľmi vysoká. To výrazne znižuje potrebu prechádzať celý zoznam detí pri každom vkladaní/vyhľadávaní, čím sa priemerná zložitosť hľadania zníži bližšie k **O(1)**.
* **Vkladanie do SinglyLinkedList:**
  + Pridanie kópie StopData do SinglyLinkedList na koniec je operácia s časovou zložitosťou **O(1)**.
* **Celková zložitosť:**
  + Časová zložitosť načítavania a budovania hierarchie je **O(n)**, kde n je celkový počet zastávok vo vstupnom súbore.

1. Tretia úroveň

**ANALÝZA ČASOVEJ ZLOŽITOSTI OPERÁCIE VLOŽ A NÁJDI:**

Pre efektívne vyhľadávanie zastávok podľa ich ID je použitá hašovacia tabuľka (ds::adt::HashTable), ktorej kľúče (StopID) sú jedinečné.

Časová zložitosť:

Priemerná: O(1)

Najhoršia: O(n) kde n je počet prvkov v tabuľke.

Analýza:

Obe operácie začínajú výpočtom hašu pre kľúč pomocou hašovacej funkcie a následným prístupom k príslušnej „priehradke" - **O(1).**

Riešenie kolízií: ds::adt::HashTable používa metódu reťazenia, kde každá priehradka môže obsahovať zoznam synonym (prvkov, ktoré sa hašovali na rovnaký index).

Vďaka efektívnej hašovacej funkcii a primeranej kapacite sú zoznamy synonym krátke. Vyhľadanie alebo vloženie prvku do takého krátkeho zoznamu trvá v priemere konštantný čas **O(1)**.

Najhorší prípad: Ak by teoreticky všetky prvky hašovali na ten istý index, všetky by boli v jednom dlhom zozname. Operácia by si potom vyžiadala prechod celého tohto zoznamu, čo by viedlo k lineárnej zložitosti O(n). – veľmi nepravdepodobné

**RIEŠENIE DUPLICÍT**

Hoci sú ID zastávok vo vstupnom súbore jedinečné, kód je pripravený na potenciálne duplicity:

Metóda StopHashTable::insert obaluje volanie základnej triedy do try-catch bloku.

Ak vnútorná implementácia vyhodí std::logic\_error pri pokuse o vloženie existujúceho kľúča, táto výnimka je zachytená. V takom prípade sa prebytočne alokovaná pamäť pre StopData objekt uvoľní a duplicitný vklad je ignorovaný, čím sa zabezpečí, že v tabuľke zostane len prvý záznam pre dané ID\*

\*Tento mechanizmus bol úspešne otestovaný s upraveným vstupným súborom obsahujúcim duplicitné ID, pričom program fungoval správne.

1. Zoznam v práci použitých štruktúr, ktoré boli implementované spoločne na cvičení

* ds::adt::MultiwayTree<T>

Využitie: Základ pre HierarchyBuilder a HierarchyNavigator.

* ds::adt::SinglyLinkedList<T>

Využitie: Implementuje zoznamy zastávok v rámci listových uzlov (ulíc) hierarchického stromu (NodeData::stops).

* ds::adt::HashTable<K, T>

Využitie: V triede StopHashTable pre rýchle vyhľadávanie zastávok podľa ich ID.

* ds::amt::ImplicitSequence<T>

Využitie: Slúži ako dočasný kontajner pre dáta v StopSorter, na ktorom sa vykonáva triediaci algoritmus.

* ds::adt::Sort<T> (s konkrétnou implementáciou ds::adt::QuickSort<T>)

Využitie: Poskytuje základ pre triediaci algoritmus v triede StopSorter, ktorý zabezpečuje efektívne zoradenie filtrovaných zastávok.