ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia
a informatiky

Algoritmy a údajové štruktúry 1

SEMESTRÁLNA PRÁCA

Vypracoval: **Patrik Pavlík**
Študijná skupina: **5ZYS21**
Predmet: **Algoritmy a údajové štruktúry 1**
Cvičiaci: *doc. Ing. Marek Kvet, PhD.*

Obsah

1. Zadanie práce

Zadanie 1

Načítajte údaje zo smerovacej tabuľky do vami zvolenej sekvenčnej údajovej štruktúry. Po načítaní teda bude načítaná 1 sekvenčná štruktúra.

Pre jednotlivé záznamy načítajte a evidujte nasledovné údaje:

• Adresu cieľovej siete a sieťovú masku (vo formáte adresa/maska, stĺpec prefix).

• Adresu suseda pri smerovaní do cieľovej siete (stĺpec next-hop).

• Čas prítomnosti záznamu v tabuľke – životnosť (stĺpec čas).

Naprogramujte univerzálny algoritmus, ktorý umožní spracovať tie údaje z údajovej štruktúry, ktoré spĺňajú zadaný predikát.

Pod spracovaním sa v našom prípade bude rozumieť vloženie údajov do inej sekvenčnej priamej štruktúry. Naplnenú priamu sekvenčnú štruktúru umožnite vypísať. Budeme využívať predikáty, ktoré vrátia pravdu, ak:

• **matchWithAddress**: zadaná IP adresa sa zhoduje s adresou cieľovej siete v prvých N bitoch

zľava podľa sieťovej masky. Príklad je na nasledujúcom obrázku.

• **matchLifetime**: životnosť záznamu v smerovacej tabuľke je v zadanom časovom rozmedzí.

Prvú úroveň teda môžeme konkretizovať na: Naprogramujte algoritmus, ktorý vyfiltruje záznamy zo smerovacej tabuľky, ktorých {adresa cieľovej siete sa v prvých N bitoch zľava podľa sieťovej masky zhoduje so zadanou adresou}/{životnosť je v zadanom rozmedzí}, do inej sekvenčnej údajovej štruktúry, ktorú potom vypíšete. Dbajte na čo najväčšiu univerzálnosť algoritmu (bude sa využívať aj neskôr).

Zadanie 2

Doplňte a/alebo upravte načítané údaje zo smerovacej tabuľky tak, aby ste **vybudovali hierarchiu IP adries podľa oktetov**.

Implementujte iterátor hierarchie IP adries v smerovacej tabuľke. Iterátor musí byť možné ručne (zadaním voľby z terminálu resp. stlačením tlačidla v aplikácii s grafickým rozhraním) presunúť (ak je to vzhľadom na jeho aktuálnu pozíciu možné) na:

• nadradenú úroveň hierarchie (teda z úrovne 2. oktetu je možné presunúť na úroveň 1. oktetu) alebo na zvoleného syna aktuálneho vrcholu hierarchie.

Uvažujme hierarchiu s koreňom totožným s aktuálnym vrcholom iterátora. Umožnite nad touto hierarchiou spustiť algoritmus z úrovne 1 s jedným z dvoch predikátov – voľbu zadá používateľ (bude sa teda možné napr. navigovať postupne z úrovne smerovacia tabuľka na IP adresy s 1. oktetom = 3 a 2. oktetom = 172 a následne nechať vypísať všetky záznamy, ktoré sú staršie ako 2 týždne a následne sa vrátiť naspäť na úroveň s 1. oktetom (v tomto príklade rovnom 3) a vypísať všetky záznamy, v ktorých sa adresa cieľovej siete v prvých N bitoch zľava podľa sieťovej masky zhoduje s 3.160.0.0).

Zadanie 3

Umožnite efektívne vyhľadať informácie o záznamoch v smerovacej tabuľke, ktorých „next-hop“ bude zadaný. Aby ste to dosiahli pridajte do vašej aplikácie tabuľky1 uchovávajúce ukazovatele na načítané záznamy z úrovne 1. Všetka doterajšia funkčnosť musí ostať zachovaná!

Zadanie 4

Pomocou algoritmu z prvej úrovne a pomocou užívateľom zadaného predikátu (z prvej úrovne) získajte zo zvoleného vrcholu hierarchie (do vrcholu sa môžete navigovať pomocou riešenia z druhej úrovne) sekvenčnú priamu údajovú štruktúru.

Naprogramujte univerzálny triediaci algoritmus, ktorý umožní zoradiť údaje z údajovej štruktúry, na základe komparátora. Budeme využívať **komparátory**, ktoré budú porovnávať:

• **comparePrefix**: IP adresa v prefixe prvého záznamu je nižšia ako IP adresa v prefixe druhého záznamu. Ak sú IP adresy v prefixoch rovnaké, tak sa porovná, či je maska v prefixe prvého záznamu menšia ako maska v prefixe druhého záznamu.

• **compareTime:** čas prítomnosti prvého záznamu je nižší ako čas prítomnosti druhého záznamu. Po zoradení vypíšte zoradené údaje spolu s hodnotami, na základe ktorých boli triedené.

2. Návrh aplikácie

Aplikácia sa skladá z viacerých tried, pričom hlavnou triedou je trieda Program, z ktorej vytvoríme inštanciu v triede main.cpp. Trieda Program obsahuje v atribútoch údajové štruktúry určené na uchovávanie RT záznamov podľa jednotlivých úrovni zadania semestrálnej práce.

Po zavolaní konštruktora sa spustí program aplikácie. Najskôr inicializujeme používané údajové štruktúry, zadáme názov súboru a pomocou triedy ktorej načítame vstupné dáta z príslušného súboru načítame dáta do implicitnej sekvencie. Kódovanie súborov bolo upravené na Windows-1250 aby následné vypisovanie na konzolu bolo zo slovenskou diakritikou. Používateľ dokáže s aplikáciou komunikovať prostredníctvom konzolového rozhrania.

2.1. Reprezentácia RT záznamu

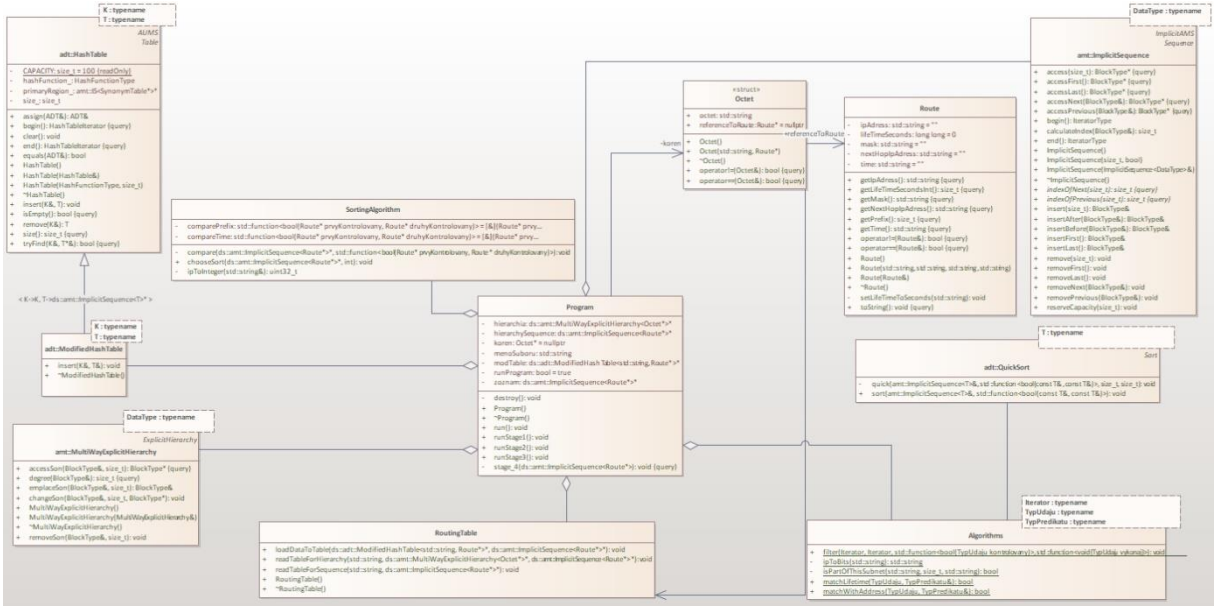
Tento RT záznam reprezentuje trieda Route do ktorej sú uložené potrebné dáta z CSV súboru.

2.2. Načítanie údajov do aplikácie

Na načítanie vstupných údajov slúži trieda RoutingTable(route_table_reader.h), ktorá obsahuje metódy na načítanie údajov do sekvencie, hierarchie a tabuľky. V triede Program si inicializujeme potrebné údajové štruktúry. V deštruktore triedy tieto objekty musíme z haldy uvoľniť aby nevznikli memory leaky.

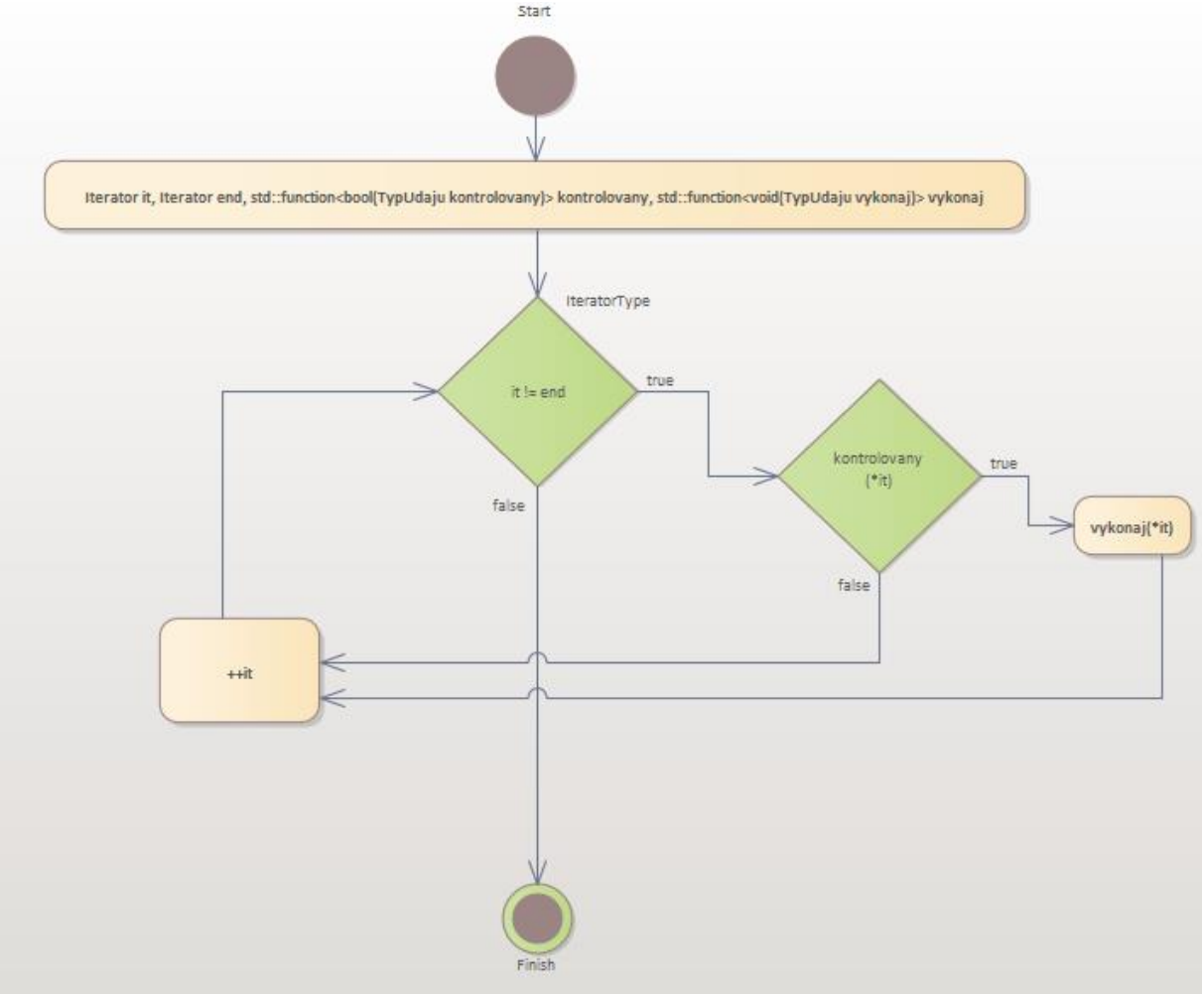
3. UML diagramy

UML diagram tried



Obrázok 1. UML diagram tried predstavujúcich program na spracovanie RT záznamov

UML diagram aktivít



Obrázok 2. Diagram aktivít univerálneho algoritmu

4. Rozbor vhodnosti použitia zvolených údajových štruktúr

4.1. Prvá úroveň

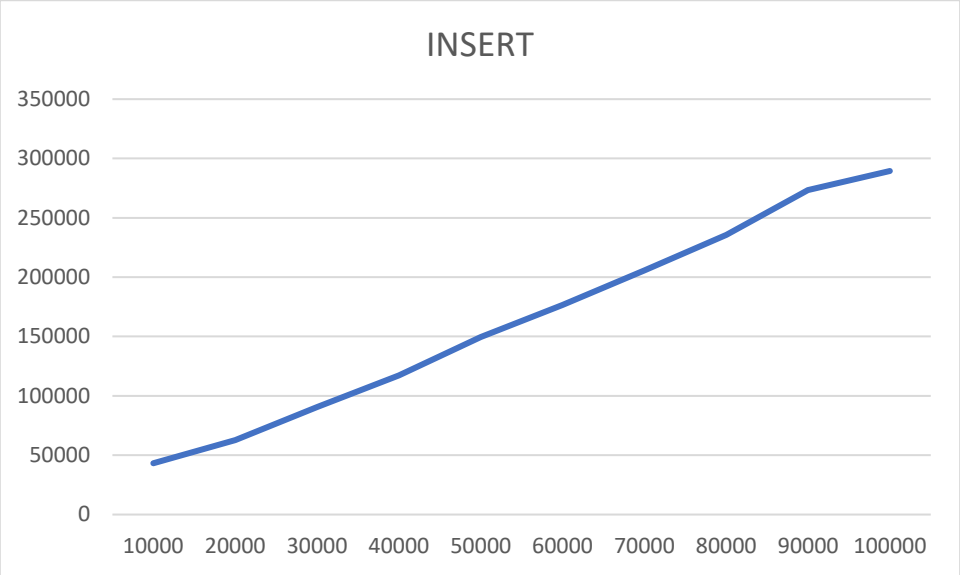
V úlohe 1 sme použili implicitnú sekvenciu kvôli jej O(1) náročnosti pri pridávaní prvkov na koniec a rýchlemu vyhľadávaniu (jednoduché získanie nasledovníka). Použitie zahŕňalo načítanie dát zo súboru a ich uloženie do sekvencie. Následne môžeme prejsť sekvenciu s použitím predikátu a vypísať hodnoty, ak predikát vrátil pravdu. K tomuto zadaniu sme si zároveň vytvorili univerzálny algoritmus, ktorý dokáže vypísať smerovacie tabuľky na základe zvoleného predikátu.

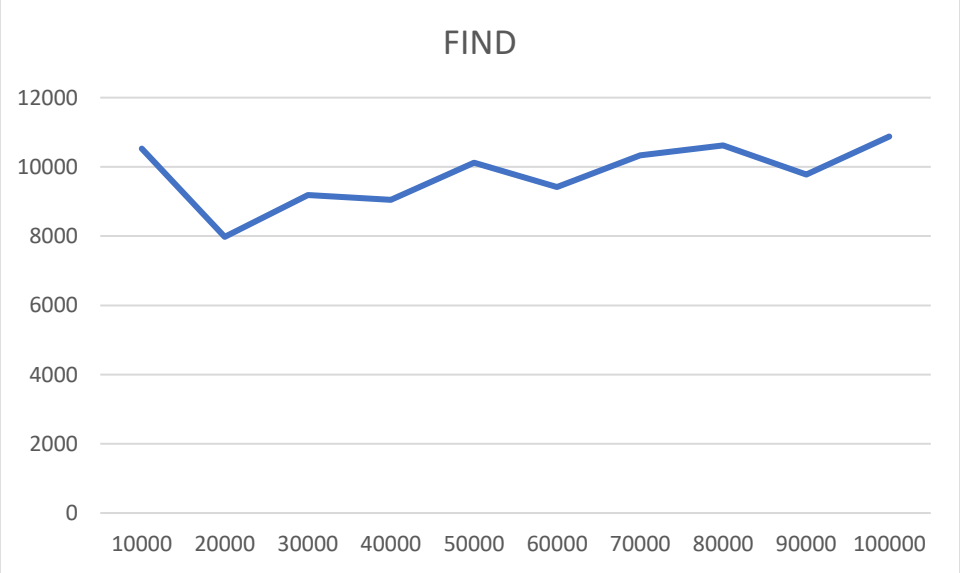
4.2. Druhá úroveň

Na načítanie routovacích záznamov do hierarchie používame viaccestnú explicitnú hierarchiu (MultiWayExplicitHierarchy). Tento typ hierarchie je nevyhnutný, pretože počet potomkov každého uzla sa líši, čo znemožňuje použitie iného typu hierarchie. Univerzálny algoritmus z prvej úrovňe je možné aplikovať aj na túto hierarchiu a výsledky následne ukladať do jednej implicitnej sekvencie (ImplicitSequence).

4.3. Tretia úroveň

Na tretej úrovni chceme mať efektívny prístup k smerovacím tabuľkám podľa ich next-hopu. Rozhodli sme sa použiť hashovaciu tabuľku (HashTable). Teoreticky je hashovacia tabuľka najrýchlejším riešením, pretože operácie vkladania a vyhľadávania majú časovú zložitosť O(1).





4.4. Štvrtá úroveň

V štvrtej úrovni sme vytvorili algoritmus pre triedenie pomocou algoritmu QuickSort z DataStructures. Triediaci algoritmus pracuje na podobnom princípe ako univerzálny algoritmus zo zadania 1. Rozdiel je v tom, že pred tým ako vypíše údaje tak ich najprv zoradí. Robí to tak, že si zoberie sekvenciu z filtrovaných predikátov, ktoré sa majú utriediť. Potom zavolá QuickSort nad sekvenciou a následne vypíše prvky danej sekvencie utriedené podľa nami zvoleného komparátora.

5.Príručky

5.1. Programátorská príručka

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

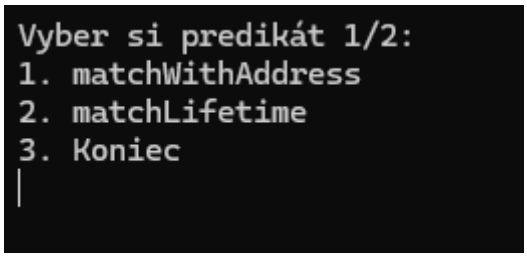
- pár iterátorov ukazujúci na začiatok a koniec položiek údajovej štruktúry (it, end),
- predikát, ktorý je vo forme lambda funkcie,
- lambda funkciu, ktorá sa vykoná pre každú filtrovanú položku ak spĺňa zadaný predikát.

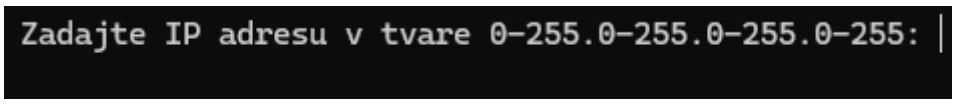
Metóda má tri šablónové parametre, **Iterator** - vďaka ktorému je možné použiť iterátor ľubovoľného typu, **TypUdaju** - vďaka ktorému je možné filtrovať položky ľubovoľného typu a **TypPredikatu** - vďaka ktorému je možné použiť lubovolný typ podla ktorého sa bude filtrovať.

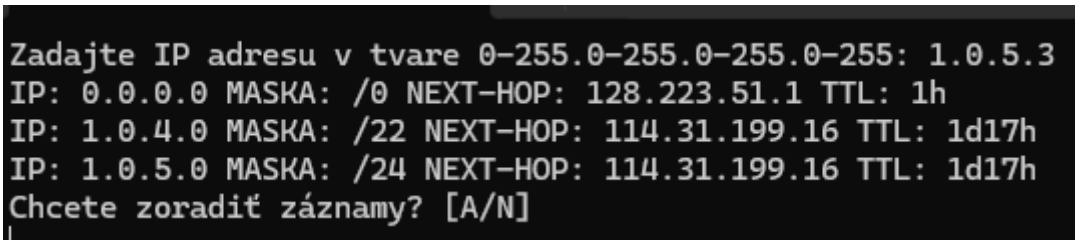
5.2. Používateľská príručka

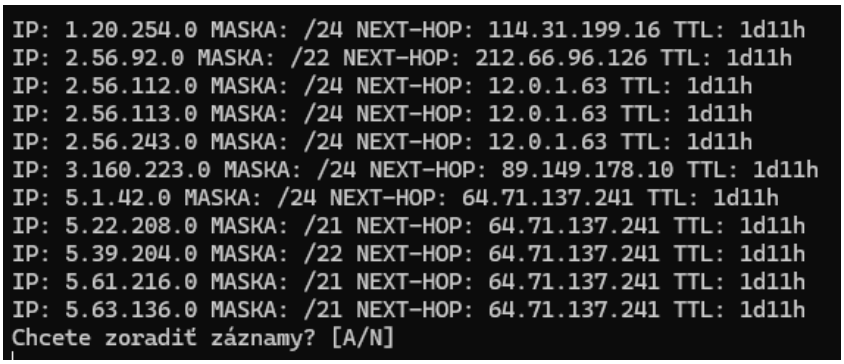
Aplikácia poskytuje konzolové rozhranie. Po zapnutí aplikácie sa zobrazí hlavné menu, v ktorom si používateľ môže zadaním príslušného čísla vybrať:

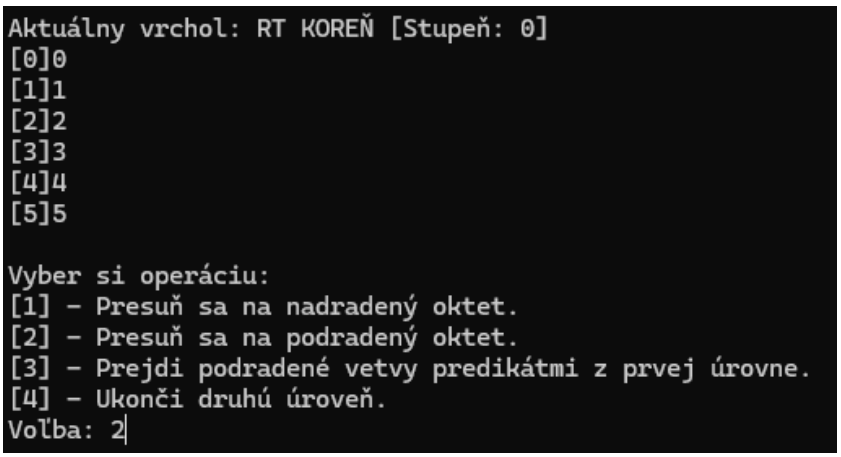
- filtrovanie RT záznamov (SP úroveň 1 + 4)
- navigácia v RT záznamoch (SP úroveň 2 + 4)
- vyhľadávanie informácií podľa názvu (SP úroveň 3)
- zmena súboru
- ukončiť aplikáciu

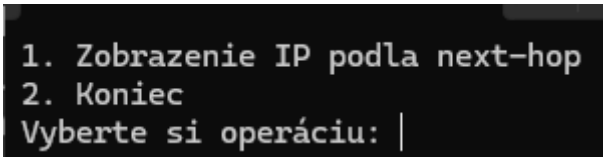












6. Implementácie jednotlivých úrovní

6.1. Implementácia prvej úrovne SP

Do našej zvolenej implicitnej sekvenčnej údajovej štruktúry (`ImplicitSequence`) sme načítali údaje spôsobom opísaným v sekcií Návrh aplikácie. Používateľa vyzveme na zvolenie predikátu, podľa ktorého si vyžaduje filtrovanie smerovacích tabuliek a tiež ho vyzveme na zadanie vyhľadávacieho argumentu (IP adresy alebo life-time(v sekundách), na ktorý má smerovacia tabuľka začínať).

6.2. Implementácia druhej úrovne SP

V druhej úrovni umožňujeme používateľovi pohyb v hierarchií smerovacích tabuliek pomocou metód accessParent(*vetva) a accessSon(*vetva,indexSyna) z `MultiWayExplicitHierarchy.`

Umožňuje sa presunúť na otca, ak aktuálna vetva nie je koreň tak sa presunie na rodiča inak sa nič nestane. Umožňuje sa presunúť na ľubovoľného syna aktuálneho vrcholu ak, aktuálny vrchol má synov, tak sa presunie na nami zvoleného syna, ak nemá žiadnych synov tak sa nič nestane.

Aby sme boli schopný spustiť univerzálny algoritmus nad takouto podhierarchiou musíme nastaviť začiatok PreOrderHierarchyIterator na aktualnyVrchol a následne môžeme spustiť, univerálny algoritmus z prvej úrovne

6.2.1. Spôsob načítavania údajov do hierarchie

Za načítavanie smerovacích tabuliek do hierarchie je zodpovedná trieda RoutingTable a struct Octet. Na začiatku vloží do hierarchie ako koreň Octet s názvom "RT KOREŇ", ktorý predstavuje pomyslený koreň hierarchie. Ďalej, ako v prvej úrovni sa postupne vložia záznamy dáta do implicitnej sekvencie, a následne sa rozdelené oktety, ktorých je spolu 5, kde piaty oktet je prefix(maska) ktorá odkazuje na vložený záznam z implicitnej sekvencie.

Oktety sa do hierarchie prideľujú týmto spôsobom:

- Na začiatku keď ešte nemá daný koreň v jeho stupni žiadnych synov tak skontroluje či je počet jeho synov rovný 0, ak áno indexy oktetov budú nastavené na nulu, následne do koreňa pridá syna, nastaví mu hodnotu 1 časti oktetu a zväčší index o 1.
- Ak sa už v danom koreni, nachádza nejaký syn tak najskôr prehľadá jeho aktuálnych synov, a kontroluje či sa už v hierarchii nachádza oktet s rovnakou hodnotou ako aktuálny ktorí sa ide vložiť, ak sa nájde tak sa ukazovateľ nastaví na nasledovníka a cyklus skončí. Ak sa nenájde, vytvorí sa nový syn, indexy 2-5 budú nastavené na nulu, pripočíta sa index syna o 1
- Keď je v piatom oktete tak robí takmer to iste, ale pri vytvorení oktetu, pridelí aj referenciu na objekt zo sekvencie

6.3. Implementácia tretej úrovne SP

V tretej úrovni umožňujeme používateľovi zadať next-hop IP adresu a aplikácia mu efektívne sprístupni informácie o dannom routovacom zázname. Využívame pri tom upravenú tabuľku s rozptýlenými záznamami(ModifiedHashTable) ktorá je potomkom tabuľky s rozptýlenými záznamami, kde sme zdedili metódu insert v ktorej sú ošetrené duplicity tak, že ak nájde už existujúci kľúč v tabuľke tak údaj vloží do existujucej IS, ak nenájde tak vytvorí novú sekvenciu kde sa budú ukladať dáta s rovnakým klúčom, a prídá údaj do IS.

6.4. Implementácia štvrtej úrovne SP

Táto úroveň pracuje s vyfiltrovanými RT záznamami z druhej úrovne, ktoré sú usporiadané v implicitnej sekvencii. Triedenie týchto záznamov zabezpečuje trieda SortingAlgorithm. Obsahuje univerzálny triediaci algoritmus, ktorý je implementovaný metódou chooseSort. Táto metóda prijíma dva parametre: implicitnú sekvenciu a komparátor vo forme lambda funkcie, ktorá definuje pravidlá pre porovnávanie jednotlivých záznamov.

7.Záver

Údajové štruktúry ktoré boli vypracované na cvičeniach predmetu Algoritmy a údajové štruktúry 1 boli použité v semestrálnej práci.

Boli použité nasledovné údajové štruktury:

- **ImplicitSequence**
- **HashTable ➔ ModifiedHashTable**
- **Multiway Explicit Hierarchy**
- **Quick sort**