A black background with white text

Description automatically generated

Algoritmy a údajové štruktúry 1

Semestrálna práca

Meno: Lukáš Beďač

Štud. skupina: 5ZYI34

Cvičiaci: Ing. Michal Varga, PhD.

Cvičenie: Štvrtok 8:00

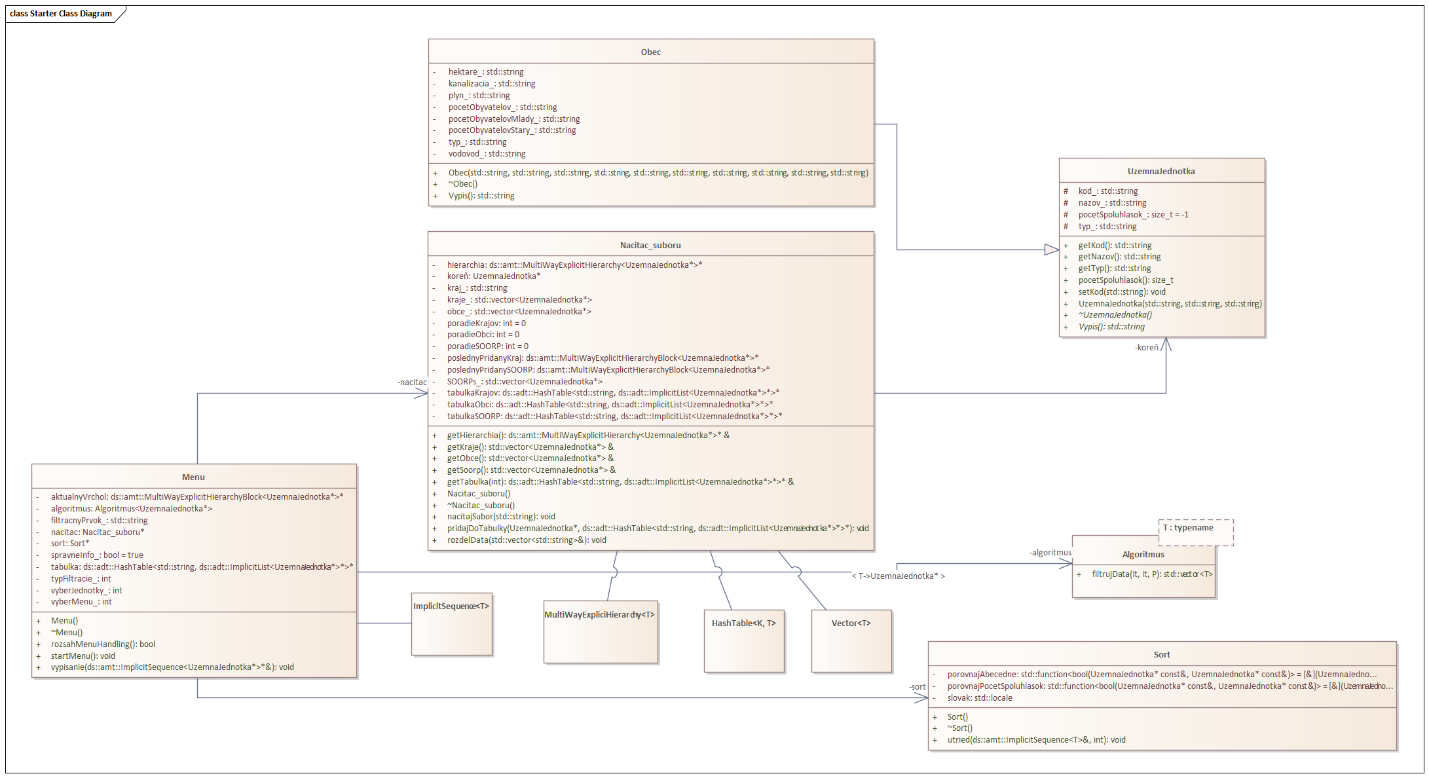
Školský rok: 2023-2024

# Návrh aplikácie

Hlavnou triedou v semestrálnej práci je trieda menu, ktorá postupne zavolá potrebné triedy pre rôzne časti program. Vo vstupnom bode program, čo reprezentuje *main*, sa zavolá trieda menu, v ktorej sa ako prvá funkcia vytvorí trieda načítač súboru, do ktorej sa aj hneď dostaneme a načítajú sa dáta. V triede načítač súboru sa načítajú dáta pre všetky úrovne naraz, pričom na ich ďalšie použitie a spracovanie používame *gettre*.

Používateľ vie komunikovať s aplikáciou prostredníctvom konzolového rozhrania. Na interakciu s aplikáciou a užívateľom slúži trieda menu.

# UML diagram tried



Obrázok 1 – UML diagram tried

*A screenshot of a computer

Description automatically generated*

Obrázok 2 – UML diagram načítač\_súboru

*A screenshot of a computer menu

Description automatically generated*

Obrázok 3 – UML diagram menu

*A screenshot of a computer

Description automatically generated*

Obrázok 4 – UML diagram územných jednotiek

*A screenshot of a computer

Description automatically generated*

Obrázok 5 – UML diagram algoritmus a sort

# Rozbor vhodnosti použitia údajových štruktúr

## Prvá úroveň

Na načítanie a uchovanie územných jednotiek používame *vectory*, pre každý typ územnej jednotky jeden, teda *vector* krajov, soorp a obcí. *Vector* používam vzhľadom na to, že v prvej úrovni to bolo povolené a v dobe vypracovávania úrovne sme nemali implementovaný kód a ani sme neboli na takej úrovni, aby sme to zvládli sami.

*Vectory* sa napĺňajú okamžite pri čítaní dát a to po každom prečítaní riadku sa rozhodne, čo je daná územná jednotka a kam sa pridá. Počas behu sa *vectory* nemodifikujú. Pre časovú obmedzenosť som prvú úroveň nemenil na *ImplicitList*.

Na filtrovanie údajov som vytvoril vlastnú triedu, v ktorej sa po vyfiltrovaní uložia dané jednotky do jedného spoločného *vectoru*.

Použitie *vectorov* v tejto časti je časovo efektívne, pretože *vector* má pri náhodnom pristúpení zložitosť O(1), pri vložení alebo vyberaní na konci O(1) a pri vložení alebo vybraní na pozícií je O(n).

## Druhá a štvrtá úroveň

Na načítanie územných jednotiek do hierarchie používame viaccestnú explicitnú hierarchiu. Túto hierarchiu používame vzhľadom na to, že ako aj v reálnom svete sú územné jednotky rozdelené podľa úrovní, tak aj v aplikácií potrebujeme takúto štruktúru. Tak isto, nemôžeme použiť k-cestnú hierarchiu, pretože každý kraj má iný počet soorp a ten má iný počet obcí, teda by sme plytvali miestom-pamäťou.

Algoritmus z prvej úrovne sa dá použiť aj pri hierarchií a tak isto sa vracia *vector* vyfiltrovaných dát, pričom sa dá použiť aj pridaný *hasType*.

Ako problém je použitie *vectoru* v 4. časti, pretože triediace algoritmy potrebujú implicitnú sekvenciu.

Na efektívne náhodné pristupovanie bude použitá implicitná sekvencia, pretože triedenie funguje na používaní náhodných (vypočítaných) indexoch, kde *ImplicitSequence* dosahuje O(1).

## Tretia úroveň

V tretej úrovni chceme pristupovať k územným jednotkám podľa názvu rýchlo a efektívne pričom musíme riešiť aj synonymá. Z tohto dôvodu som sa rozhodol použiť *HashTable* – tabuľka s rozptýlenými záznamami. Teoreticky je táto tabuľka najrýchlejšia pri operáciách vlož a nájdi, pretože majú zložitosť O(1) ale len ak používame vhodnú *hashovaciu* funkciu.

# Používateľská príručka

Na rozhranie aplikácie som sa rozhodol použiť konzolové rozhranie. Po zapnutí aplikácie sa používateľovi zobrazí hlavné menu, v ktorom sa nachádza:

* 1. časť semestrálnej práce
* 2. a 4. časť semestrálnej práce
* 3. časť semestrálnej práce
* Ukončenie programu

## Prvá časť – úroveň 1

Túto časť si zobrazíte po zvolení možnosti 1 v hlavnom menu. Po zobrazení úrovne 1 sa vám zobrazí ako možnosť zvoliť si filtrovanie: 1.krajov, 2.soorp alebo 3.obcí, alebo 4. ukončiť program.

Po zvolení 1., 2. alebo 3. možnosti sa vám zobrazí výzva na napísanie znakov, ktoré chcete vyhľadať, pričom po napísaní znakov dostane 2 možnosti filtrovania a to: 1. Jednotka začína na: a 2. Jednotka obsahuje znaky: . Po zvolení jednej s možností sa vám v konzolovom rozhraní vypíšu vyfiltrované územné jednotky s ich ďalším popisom.

## Druhá a štvrtá časť – úroveň 2 a 4

Nasledujúcu časť si zobrazíte zvolením možnosti 2 v hlavnom menu. Po zvolení sa vám zobrazí informácia o vrchole, v ktorom sa v hierarchii nachádzate a možnosti pohybu v nej. Možnosti sú:

1. Pokračovať hlbšie do hierarchie:   
   Po zvolení danej možnosti sa vám vypíšu synovia vrcholu, v ktorom sa nachádzate, kde po vypísaní dostanete výzvu, na ktorého syna sa chcete posunúť, čo sa stane vašim aktuálnym vrcholom. Postupne pri tejto možnosti sa dostanete až k obciam, kde synovia nie sú a teda vypíše sa upozornenie že sa nedá ísť hlbšie.
2. Vrátiť sa na otca:  
   Po zvolení možnosti 2, ak sa nachádzate v koreni hierarchie, dostanete upozornenie že sa nedá posunúť vyššie, pretože ste v koreni. Ak sa nachádzate kdekoľvek pod koreňom, teda vás aktuálny vrchol nie je Česká republika, táto možnosť vás dostane k otcovi syna, v ktorom sa nachádzate.
3. Spustiť prehľadávanie podľa zadaného typu:  
   Po zvolení nasledovnej možnosti sa dostanete k ďalšiemu menu, ktoré obsahuje 3 položky, podľa ktorých môžete filtrovať. Sú to:

* 1. Jednotka začína na:
* 2. Jednotka obsahuje znaky:
* 3. Jednotka je typu:

Prvé 2 možnosti sa nijako nelíšia filtrovaním od úrovne 1, len zadáte názov až po zvolení filtrácie. Posledná možnosť je nová a po jej zvolení si môžete vybrať filtrovanie podľa krajov, soorp alebo obcí z daného vrcholu nižšie.

1. Koniec programu

## Tretia časť – úroveň 3

Táto úroveň sa zobrazí po zvolení hodnoty 3 v hlavnom menu, po ktorej sa vám ukáže menu podobné ako v úrovni 1. Vyberiete si, ktoré územné jednotky chcete prehliadať v tabuľke, dostupné sú: 1. kraje, 2.soorp, 3.obce a po zvolení jednej z možností, bude výzva v konzolovom rozhraní, aby ste zadali názov, podľa ktorého chcete vyhľadávať. Vyhľadávanie v tejto časti funguje na presne definovaný názov územnej jednotky.  
Po napísaní názvu sa vám vypíšu jednotky s ich príslušnými informáciami.

# Implementácia úrovní

## Úroveň 1

Pri zapnutí aplikácie sa vám v menu zavolá inštancia triedy *načítač*\_*súboru*, ktorá načíta územné jednotky do 3 vektorov, pričom im hneď ukladá aj dané informácie o danej územnej jednotke. Keďže jediná obec má rozdielne dáta, obec je potomok triedy územná jednotka a vektory sú všetky plnené územnými jednotkami.

Na filtrovanie máme vytvorenú triedu algoritmus, do ktorej z triedy menu pri interakcií s požívateľom posielate daný predikát, pričom sa k tomu pošle aj *iterátor* – začiatok a koniec vektoru dát. Ten následne vráti vektor filtrovaných dát a tie sa okamžite vypíšu používateľovi v konzolovom rozhraní.

## Úroveň 2

Tak isto ako v úrovni 1 hneď po zapnutí aplikácie sa *náčítač*\_*súboru* rozšíril a upravil len o to, že vždy po pridaní územných jednotiek do vektora sa tak isto pridali aj do hierarchie, pričom sa ponastavovali vzťahy medzi jednotlivými vrcholmi.

Na pohyb v hierarchii na otca alebo syna sa používa len jednoduché prechádzanie synov aktuálneho vrcholu alebo jeho otca, pričom ak jeden alebo druhý vráti *nullptr,* vypíše sa upozornenie.

Na vyfiltrovanie dát sme museli pridať 2 premenné *PreOrderHierarchyIterator,* ktoré sa následne posielajú ako *iterátor* do triedy algoritmus ako začiatok a koniec. Tieto *iterátory* nám umožnia prechádzať hierarchiu od aktuálneho vrcholu až po koniec. Hierarchia od aktuálneho vrcholu a nie koreňa, vytvorí podhierarchiu a teda koreňom v nej sa stáva aktuálny vrchol a PreOrderHierarchyIterator si vymaže referenciu otca z aktuálneho vrchola, aby z neho nemohol vyjsť – teda hierarchia si vymaže sekvenciu pozícií nad aktuálnym vrcholom – koreňom, a pri prehliadaní sa nedostaneme nad náš aktuálny vrchol.

Pri načítaní jednotlivých územných jednotiek sa program správa vždy inak:  
Ak načítame kraj, spoliehame sa na to že v celom riadku *csv* súboru sa nachádza len 1 údaj a to názov kraju. Ak to tak je, vytvorí sa nový kraj bez kódu kraju a program preskočí na načítanie ďalšieho riadku, pričom sa nastaví premenná *poslednyPridanyKraj* na daný načítaný kraj, ktorý nám vráti metóda *emplaceSon*. Keď sa dostaneme na ďalší riadok v súbore, skontroluje sa či posledný pridaný kraj má kód prázdny, ak to je pravda prvý údaj z riadku sa zapíše ako kód poslednému načítanému kraju.

Ak načítame soorp, nachádzame sa už minimálne v 2. riadku *csv* súboru a tam kontrolujeme, či sa poradie obce = 1, ak áno, vieme že to je soorp a načítame novú soorp s prvými údajmi k nej až po poradie obcí.

Následne načítame obce a to tak, že po načítaní soorp vieme, že daná soorp je aj obec a vytvorí sa aj nová inštancia územnej jednotky, obec, ktorá si načíta potrebné údaje. Ak program prebehne bez predošlých kontrol, automaticky sa dostane na vytvorenie novej obce, pretože údaje nemôžu byť nič iné – spoliehame sa na správnosť súboru.

A screen shot of a computer

Description automatically generated

Obrázok 6 – ukážka prvotného menu hierarchie

*A screenshot of a computer program

Description automatically generated*

Obrázok 7 – ukážka výberu pokračovania do hierarchie a zmena aktuálneho vrcholu

## Úroveň 3

V tejto úrovni potrebujeme vytvoriť tabuľku s údajmi, ktoré sa načítajú ako aj do vektorov tak do hierarchie, teda najjednoduchšia možnosť je načítanie údajov zároveň s prvými úrovňami aj do tabuľky. Pri načítaní sa pridá už nastavená územná jednotka aj s dátami do tabuľky, pričom sa predtým musí vykonať *hashovacia* funkcia, ktorú sme si definovali.

Program obsahuje 3 tabuľky, pre kraje, soorp a obce a každá používa tú istú *hashovaciu* funkciu, len má rozdielne veľkosti. Ako veľkosť tabuliek som sa rozhodol použiť pri krajoch prvočíslo 13 aj keď ich je 14. Pri soorp som použil veľkosť 53, taktiež prvočíslo a pri obciach som použil veľkosť 3041 – prvočíslo. Po zisťovaní, ako zaručiť dobré rozptýlenie údajov v tabuľke, som vyhľadal že je dobré používať prvočísla ale kapacitu si treba zvoliť vhodne. Kapacitu som volil po konzultáciách so spolužiakmi a keďže nemám urobenú časovú zložitosť, môžeme dúfať že zvolená *hashovacia* funkcia s kapacitou sú vhodne navrhnuté.

*Hashovacia* funkcia funguje na základe názvu územnej jednotky, ktorá sa znak po znaku prekonvertuje do *ascii* hodnoty a sčítava sa. Táto hodnota sa následne použije ako hodnota *modulo* kapacita a vyjde nám index umiestnenia v tabuľke.

### Riešenie duplicitných kľúčov v tabuľkách

Keďže používame *HahTable*, je potrebné aby sme dokázali ukladať duplicitné dáta (synonymá), teda napríklad obec Adamov, ktorá sa nachádza v súbore až 3krát. Toto je zabezpečené tým, že miesto ukladania územnej jednotky do tabuľky, ukladáme implicitný list, ktorý obsahuje danú územnú jednotku. Na riešenie duplicít bola použitá nami vytvorená metóda v triede *načítač*\_*súboru* a to *pridajDoTabulky,* kde parametre sú územná jednotka, ktorú chceme vložiť do tabuľky a tabuľka, teda *HashTable.* V tejto metóde skontrolujeme pomocou metódy *tryFind*, či sa už daná jednotka nachádza v tabuľke, ak nám vráti metóda pravdu, vráti sa list s danou jednotkou a do neho vložíme našu jednotku, tým pádom máme v liste 2 jednotky s rovnakým kľúčom. Ak nám metóda ale vráti nepravdu, musíme si dynamicky vytvoriť nový list, vložiť doňho jednotku a jednotku uložiť do tabuľky podľa vypočítaného indexu.

## Úroveň 4

V tejto úrovni musíme dosiahnuť utriedenie už vyfiltrovaných jednotiek z druhej úrovne. Keďže od úrovne 1 používame vector*<*UzemnaJednotka*\*>,* nastáva problém že triediace algoritmy nám túto štruktúru nezoberú ako vstupný parameter. Preto sme po odfiltrovaní pridali lokálnu premennú typy *ImplicitSequence,* do ktorej po odfiltrovaní dát, dáta prekopírujem, resp. vložím do sekvencie ukazovatele na dáta vo *vectore.* Týmto sme dosiahli, že môžeme využiť triediaci algoritmus a rozhodli sme sa použiť *QuickSort,* čisto z vlastného záujmu. Na použitie triediaceho algoritmu sme vytvorili novú triedu *sort,* v ktorej sú vytvorené 2 *komparátory*, ktoré sa vyberajú podľa toho, ako chce užívateľ triediť dáta. V konzolovom rozhraní, používateľ zadá index triediacej metódy a zavolá sa metóda utrieď, kde sa pošle ako parameter *ImplicitSequence* a index *komparátora*. Po tom, ako sa zvolí index sa zavolá metóda sort z *QuickSortu* a vykoná sa utriedenie.

Trieda obsahuje 2 premenné, ktoré reprezentujú *komparátory* pre *QuickSort* a to sú:

1. Abecedné utriedenie
2. Utriedenie podľa počtu spoluhlások

V triede používame aj inštanciu *std*::*locale*, ktorá nám umožňuje porovnávať aj písmenká s diakritikou korektne.

# Záver

Na záver by som povedal, že aplikáciu som implementoval podľa mojich predstáv, až na jednu jedinú vec a to je prerobenie vectorov z prvej časti na *ImplicitList,* ale po konzultácií mi bolo povedané že sa tým nemusím starať. Semestrálnu prácu hodnotím veľmi pozitívne, zmenila mi pohľad na používanie štruktúr v aplikáciách, ktorú som po celú dobu tohto semestra využíval aj pri bakalárskej práci.

Použité štruktúry z cvík: ImplicitSequence, HashTable, MultiWayExplicitHierarchy, ImplicitList