Obsah obrázku text

Popis byl vytvořen automaticky

**Semestrálna práca z AUS**

(Dokumentácia)

Meno: Dávid Mičo

Študijná skupina: 5ZYI36

Akademický rok: 2022/2023

Semester: Letný

**Obsah**

[Návrh aplikácie 3](#_Toc135604829)

[UML diagram tried 4](#_Toc135604830)

[Rozbor vhodnosti použitia údajových štruktúr 5](#_Toc135604831)

[Používateľská príručka 6](#_Toc135604832)

[Prvá úroveň 7](#_Toc135604833)

[Druhá úroveň 9](#_Toc135604834)

[Tretia úroveň 10](#_Toc135604835)

[Štvrtá úroveň 11](#_Toc135604836)

# Návrh aplikácie

Moju aplikáciu som si rozdelil do viacerých úrovní. Algoritmy týkajúce sa prvej, druhej a tretej úrovni sú poväčšine implementované v triede Algoritmus. Nachádzajú sa tu metódy na načítavanie údajov z csv súborov a rôzne algoritmy potrebné pre prvé tri úrovne SP. Načítavanie dát v druhej a tretej úrovni SP sa realizuje v jednej spoločnej metóde. V mojom projekte sa nachádza jedna main trieda s názvom SemestralnaPraca.cpp. V tejto triede sú implementované štyri main funkcie, medzi ktorými sa môžeme prepínať pomocou preprocesora. Implementácia štvrtej úrovne SP, samotný triediaci algoritmus je definovaný v samostatnej triede s názvom TriediaciAlgoritmus. Jednotlivé územné jednotky sú definované pomocou spoločného predka s názvom UzemnaJednotka, ktorej potomkovia sú triedy Kraj, Okres a Obec. Potomkovia sa líšia od tohto predka tým že majú definovaný konkrétny typ a majú inak implementované metódy toString.

# UML diagram tried

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

Automaticky generovaný popis

# Rozbor vhodnosti použitia údajových štruktúr

**Prvá úroveň**

Na uchovávanie všetkých územných jednotiek určitého typu som si vybral Implicitnú sekvenciu lebo Implicitná sekvencia zaberá v pamäti menej miesta ako Explicitná(jej blok uchováva ešte smerník na nasledujúci prvok). Každú územnú jednotku vkladám vždy na koniec čím zložitosť vkladania je v mojom prípade O(b) – kde b je n/B (n je počet prvkov a B je veľkosť buffra) čo znamená že typický prípad zložitosti pri vkladaní je O(1) až na prípad keď dojde k expanzii(musíme zväčšiť kapacitu sekvencie).

**Druhá úroveň**

Pri implementácii hierarchie naplnenej územnými jednotkami som si zvolil Viaccestnú explicitnú hierarchiu, pretože v tomto prípade sa mi hodí(jednotlivé vrcholy majú rôzny počet synov). Do tohto typu sa dajú jednoducho vkladať nové vrcholy(hierarchia využíva vhodné operácie sekvencie, ktorá ukladá synov. Zložitosť vkladania nového syna je v mojom prípade O(1) pretože ho vkladám vždy na koniec sekvencie, kde sú uložení synovia daného vrcholu.

**Tretia úroveň**

Na uchovávanie všetkých územných jednotiek určitého typu v tabuľkách s kľúčom som si vybral Utriedenú sekvenčnú tabuľku, pretože jej výhoda je že kľúče v nej budú pri výpise zoradené(v našom prípade podľa názvu), čím už nemusíme tieto výstupy zoraďovať. Kvôli tomu má vkladanie do tejto štruktúry zložitosť O(n). Zložitosť operácie nájdi je v tejto tabuľke menšia ako pri iných sekvenčných tabuľkách – O(log(n)) vďaka tomu že táto operácie je naimplementovaná s využitím efektívneho polenia intervalov.

# Používateľská príručka

Moja aplikácia obsahuje štyri metódy main v súbore SemestralnaPraca.cpp. Prepínanie medzi nimi je umožnené pomocou preprocesora. Funguje to tak, že keď chcete napríklad spustiť aplikáciu prvej úrovne, tak odkomentujete riadok #define UROVEN1 a ostatné riadky so slovíčkom define zakomentujete. Pri spustení projektu sa vám potom automaticky spustí táto úroveň. Vstupy sa zadávajú na konzolu pomocou klávesnice. Podľa pokynov si volíte možnosti ktoré poskytuje daná úroveň zadaním voľby(číslo). Aplikáciu môžete ukončiť počas behu podľa pokynov(zadaním jednotky alebo písmena ‘k’, prípadne reťazca ‘koniec’). Pri funkciách startsWithString a containsString nie je vyhľadávanie dát s daným kritériom case sensitive a možete zadávať aj reťazce obsahujúce medzeru.

# Prvá úroveň

Obrázok, na ktorom je text, diagram, rad, plán

Automaticky generovaný popis**Diagram aktivít univerzálneho algoritmu**

**Programátorská príručka**

Do môjho univerzálneho algoritmu sa posielajú dva ľubovoľné iterátory(zaciatok a koniec sekvencie, ktorú chcem filtrovať) a predikát(ako lambda funkcia). Ak chceme pridať do programu ďalší predikát tak jednoducho vytvoríme lambda funkciu s týmto predikátom a pošleme ju do tohto algoritmu. Možnosť poslania ľubovoľného typu iterátorov zabezpečuje šablóna funkcie, ktorá je nad touto funkciou definovaná.

# Druhá úroveň

V tejto úrovni sme mali efektívne načítať územné jednotky ako hierarchiu reprezentujúcu celé Slovensko, na základe ich atribútu note a sort code. Ja načítavam do hierarchie kraje, okresy a obce tak že si najprv otvorím csv súbor s krajmi, v cykle ich povkladám do hierarchie ako synov koreňa(slovenská republika), potom si otvorím csv súbor s okresmi a vypýtam si od koreňa množinu jeho synov, tú postupne prechádzam a vkladám do týchto vrcholov príslušné okresy. Nakoniec si otvorím posledný csv súbor s obcami a vo vonkajšom cykle prechádzam jednotlivé kraje a vo vnútornom prechádzam jednotlivé okresy týchto krajov a vkladám do nich príslušné obce. Každý riadok príslušného csv súboru prechádzam len raz, čiže zložitosť načítavania je O(n).

# Tretia úroveň

V tejto úrovni sme mali povkladať všetky územné jednotky do tabuľky obsahujúcej ich typ. V triede Algoritmus mám definované tri tabuľky, jedna na kraje, druhá na okresy a tretia na obce. Na implementáciu tejto úrovne som si zvolil Utriedenú sekvenčnú tabuľku. Kolízie som riešil pomocou zreťazovania. To znamená že som si synonymá vkladal postupne do implicitnej sekvencie, čiže kľúč v mojej tabuľke predstavuje oficiálny názov územnej jednotky a hodnota je Implicitná sekvencia obsahujúca všetky územné jednotky s rovnakým oficiálnym názvom.

# Štvrtá úroveň

V štvrtej úrovni sme triedili výstupy funkcií z predchádzajúcich úrovní pomocou dvoch spôsobov(abecedne a pomocou počtu samohlások). Pri triedení pomocou počtu samohlások vypisujem na konzolu ku každej územnej jednotke aj počet samohlások v jej názve. Na triedenie som použil triediaci algoritmus QuickSort lebo má lepšiu zložitosť ako triediace algoritmy s priamym vkladaním. Jeho zložitosť je O(n\*log(n)).