**SEMESTRÁLNA PRÁCA 2**

**Algoritmy a údajové štruktúry 2**

**Michal Murín, 5ZIS12 2022/2023**

**Obsah**

[**1.** **Vlastná implementácia BVS** 4](#_Toc118216004)

[1.1. **Metóda Insert()** 4](#_Toc118216005)

[1.2. **Metóda Find()** 4](#_Toc118216006)

[1.3. **Metóda Delete()** 4](#_Toc118216007)

[1.4. **Výsledná podoba stromu** 5](#_Toc118216008)

[1.5. **Metóda InsertRange()** 5](#_Toc118216009)

[1.6. **Metóda BalanceTree()** 5](#_Toc118216010)

[1.7. **Metóda GetInterval(T start, T end)** 6](#_Toc118216011)

[1.8. **Metóda Iterate(TraversalType type)** 6](#_Toc118216012)

[**2.** **Aplikácia „Zdravotná karta“** 7](#_Toc118216013)

[2.1. **Návrh aplikácie** 7](#_Toc118216014)

[2.2. **Zložitosť operácií** 8](#_Toc118216015)

[ **Vyhľadanie záznamov pacienta:** 8](#_Toc118216016)

[ **Vyhľadanie pacientov v nemocnici podľa mena a priezviska:** 8](#_Toc118216017)

[ **Záznam začiatku hospitalizácie pacienta:** 8](#_Toc118216018)

[ **Ukončenie hospitalizácie:** 8](#_Toc118216019)

[ **Výpis hospitalizovaných pacientov v nemocnici v intervale** 9](#_Toc118216020)

[ **Pridanie pacienta** 9](#_Toc118216021)

[ **Vytvorenie podkladov pre účtovné oddelenie** 9](#_Toc118216022)

[ **Výpis aktuálne hospitalizovaných pacientov v nemocnici:** 9](#_Toc118216023)

[ **Výpis aktuálne hospitalizovaných pacientov, ktorí sú poistencami zadanej poisťovne** **– podľa mena:** 9](#_Toc118216024)

[ **Výpis aktuálne hospitalizovaných pacientov, ktorí sú poistencami zadanej poisťovne** – podľa rodného čísla: 10](#_Toc118216025)

[ **Optimalizácia uloženia dát:** 10](#_Toc118216026)

[ **Pridanie nemocnice** 10](#_Toc118216027)

[ **Výpis nemocníc podľa názvu** 10](#_Toc118216028)

[ **Zrušenie nemocnice:** 10](#_Toc118216029)

[**2.3.** **Grafické rozhranie** 11](#_Toc118216030)

[**2.4.** **Import a Export dát** 11](#_Toc118216031)

[**Prílohy:** 12](#_Toc118216032)

[**1.** **UML Diagram Projektu AUS2\_MurinMichal** 12](#_Toc118216033)

[**2.** **UML Diagram projektu GUI:** 13](#_Toc118216034)

1. **Vlastná implementácia BVS**

Pre vylepšenia vlastností binárneho vyhľadávacieho stromu, som si medzi atribúty triedy BSTNode (vrchol v strome), pridal atribút s názvom „Usage“, v ktorom sa uchováva počet použití, respektíve vyhľadaní daného vrcholu v strome. Pri vzniku vrcholu sa táto hodnota nastaví na 1.

Počas životnosti stromu sa musí zachovať táto vlastnosť:

* Synovia každého vrcholu majú menšiu alebo rovnakú hodnotu parametra Usage
  1. **Metóda Insert()**

Metóda vkladania prvku do stromu sa v zásade nelíši od štandardného binárneho vyhľadávacieho stromu. V mojej implementácii používam pomocnú metódu FindBSTNode(), ktorá v prípade neúspešného hľadania prvku vráti referenciu na vrchol, ktorý by mal byť otcom hľadaného prvku. V takom prípade iba jednoducho vložím prvok ako syna vráteného vrcholu.

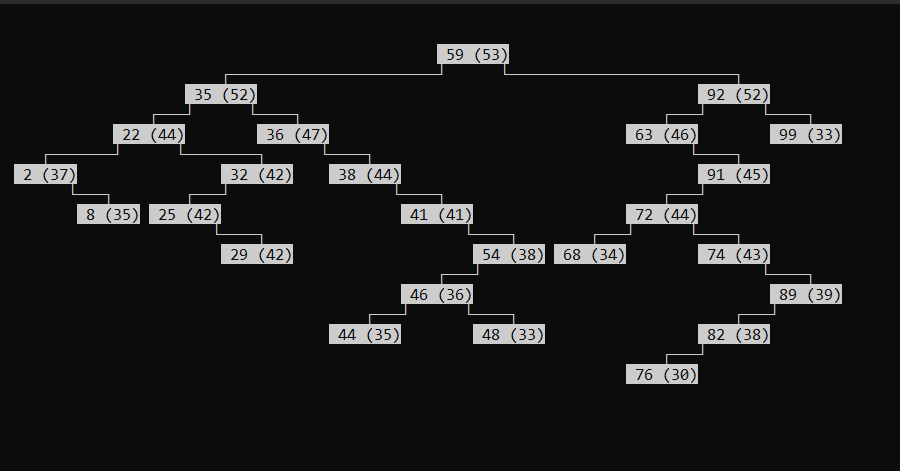
* 1. **Metóda Find()**

Metóda Find() rovnako využíva pomocnú metódu FindBST(), v tomto prípade by sa ale mal vrátiť skutočne hľadaný prvok. Po nájdení tohto prvku sa navýši jeho hodnota Usage o 1 a vrchol sa prípadne prerotuje tak, aby platila vlastnosť definovaná vyššie, pomocou metódy RebalanceUsage(BSTNode node)

* 1. **Metóda Delete()**

Pri mazaní prvku sa najskôr hľadaný prvok nájde pomocou metódy FindBSTNode(). Ak sa prvok nájde, skontroluje sa či je list – ak áno jednoducho sa vymaže, ak nie, vrchol rotuje smerom k listom až kým sa sám nestane listom a následne sa vymaže.

* 1. **Výsledná podoba stromu**

Pri splnení vyššie popísaných podmienok by mohol strom vyzerať približne nasledovne:

Strom na obrázku vznikol po generovaní náhodných volaní metódy Find() v testovacom strome s 25 vrcholmi. V zátvorke v každom vrchole je uvedená hodnota atribútu Usage.

* 1. **Metóda InsertRange()**

Pri vkladaní dávky v mojej implementácii používam pomocnú metódu GetIdealLevelOrder(), ktorá mi pomocou hľadania mediánu pre-usporiada prvky tak, v akom poradí by sa mali vkladať aby strom ostal vyvážený. Metóda sa nachádza vo vnorenej triede BaseFunctions<K>, pre jej znova použitie pri balansovaní stromu.

* 1. **Metóda BalanceTree()**

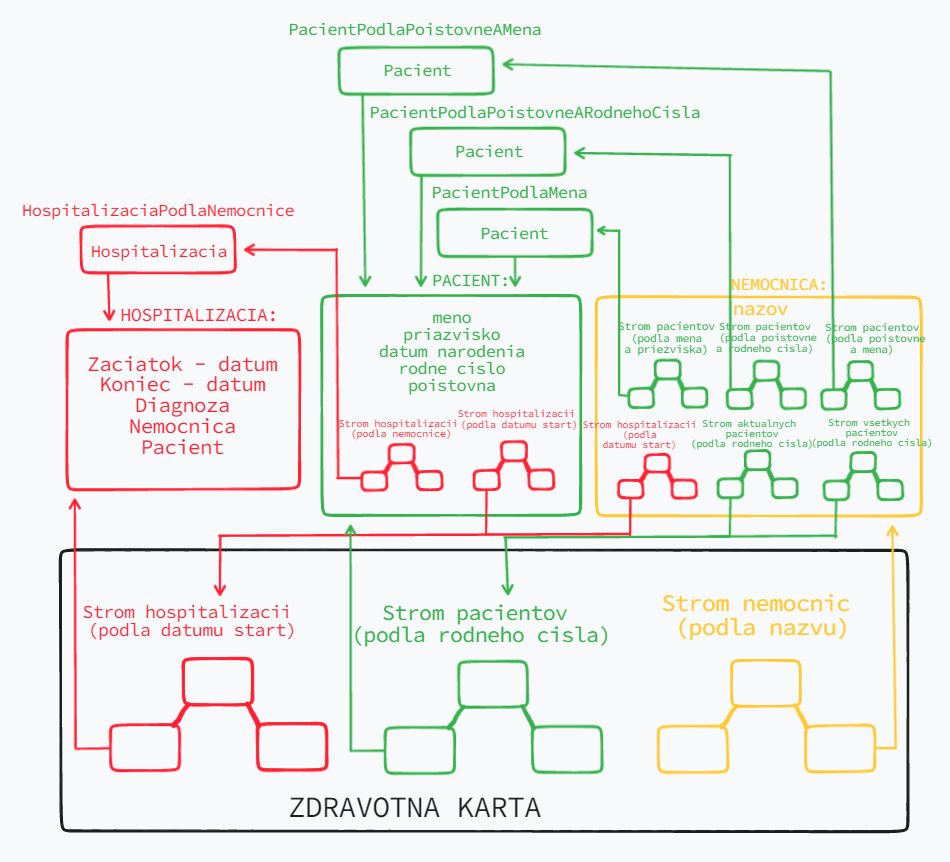
Metóda na balansovanie stromu využíva rovnakú pomocnú metódu ako InsertRange(), v tomto prípade ale potrebujeme pre-usporiadať samotné vrcholy stromu, nie len ich dáta. Po pre usporiadaní vrcholov do ideálneho poradia prehliadky Level Order, rotujeme postupne každý vrchol, počínajúc od koreňa, až kým nedosiahne svojho ideálneho rodiča. Algoritmus končí keď prerotuje predposlednú úroveň stromu. Ak celý algoritmus prebehol v poriadku, poslednú úroveň už nie je potrebné rotovať, nakoľko by mala byť už správne usporiadaná.

* 1. **Metóda GetInterval(T start, T end)**

Metóda GetInterval() hľadá dáta v strome, ktoré sa nachádzajú v zadanom intervale hodnôt. Pri prehľadávaní sa najskôr nájde začiatok intervalu (ak taký vrchol v strome neexistuje, nájde sa jeho „potencionálny“ rodič), pomocou metódy FindBSTNode(), ktorá sa používa aj pri základných metódach popísaných vyššie. Po nájdení začiatku skontrolujeme či prvok do intervalu patrí, ak áno vrátime ho a pokračujeme na jeho IN ORDER nasledovníka, až kým tento nasledovník nevypadne z intervalu.

* 1. **Metóda Iterate(TraversalType type)**

Metóda Iterate() iteruje celým stromom. To akou prehliadkou iteruje, určuje jej parameter type, typu TraversalType, čo je enum vymenovaných prehliadok stromu. Metóda následne prechádza strom určenou prehliadkou a postupne vracia dáta jednotlivých vrcholov stromu-

1. **Aplikácia „Zdravotná karta“**
   1. **Návrh aplikácie**

Jadro aplikácie sa nachádza v triede HealthCard. V tejto triede sa udržiavajú všetky dáta rozdelené do troch stromov, pre objekty pacientov, nemocníc a hospitalizácií. Ďalej sa tu nachádzajú všetky metódy potrebné na funkcionalitu popísanú v zadaní semestrálnej práce.

Trieda Patient v sebe uchováva strom referencií na hospitalizácie, zoradené podľa dátumu začiatku hospitalizácie, a strom referencií na hospitalizácie, zoradené podľa názvu nemocnice, ďalej komparátor a pomocné metódy.

Trieda Hospital udržiava stromy referencií na všetkých svojich pacientov, aktuálnych, všetkých pacientov zoradených podľa mena, podľa poisťovne a mena, a podľa poisťovne a rodného čísla. Ďalej uchováva strom referencií na hospitalizácie danej nemocnice. V triede je tiež komparátor a pomocné metódy.

Trieda Hospitalization v sebe uchováva základne atribúty, referenciu na nemocnicu ako aj pacienta.

UML diagram celej aplikácie je v prílohe.

* 1. **Zložitosť operácií**
* **Vyhľadanie záznamov pacienta:**

1. Nájdenie nemocnice podľa názvu: O(log(N)), N=počet nemocníc
2. Nájdenie pacienta podľa rodného čísla v strome nájdenej nemocnice: O(log(PN)), PN=počet všetkých pacientov nemocnice N
3. Celková zložitosť: O(log(N) + log(PN))

* **Vyhľadanie pacientov v nemocnici podľa mena a priezviska:**

1. Nájdenie nemocnice podľa názvu: O(log(N)), N=počet nemocníc
2. Nájdenie intervalu pacientov so zadaným menom: O(log(PN) + L), L=počet prvkov v intervale, PN=počet všetkých pacientov nemocnice N
3. Celková zložitosť: O(log(N) + log(PN))

* **Záznam začiatku hospitalizácie pacienta:**

1. Nájdenie pacienta: O(log(P)), P=počet pacientov
2. Nájdenie nemocnice: O(log(N)), N=počet nemocníc
3. Vloženie hospitalizácie do stromu: O(log(H)), H=počet hospitalizácií
4. Aktualizovanie stromov v pacientovi a nemocnici (do nemocnice sa pridá nová hospitalizácia, pacient, do pacienta jeho aktuálna nemocnica, hospitalizácia): O(SUM(log(xi)) cez I), xi=počet záznamov v jednotlivých stromoch, I= stromy, ktoré aktualizujeme
5. Celková zložitosť: O(log(P) + log(N) + log(H) + 8\*log(x))

* **Ukončenie hospitalizácie:**

1. Nájdenie nemocnice: O(log(N)), N=počet nemocníc
2. Nájdenie pacienta v danej nemocnici: O(log(PN)), PN=počet všetkých pacientov nemocnice N
3. Vymazanie záznamu z troch stromov nemocnice: O(SUM(log(xi)) cez I), xi je počet záznamov v danom strome, I=stromy nemocnice
4. Celková zložitosť: O(log(N) + log(PN) + 3\*log(x))

* **Výpis hospitalizovaných pacientov v nemocnici v intervale**

1. Nájdenie nemocnice: O(log(N)), N=počet nemocníc
2. Prechádzanie hospitalizácii zoradených podľa dátumu štart:
   1. Ak je interval na začiatku: O(l), l=počet prvkov v intervale
   2. Ak je interval na konci: O(HN), HN=počet hospitalizácií nemocnice N
3. Celková zložitosť: O(log(N) + HN/2)

* **Pridanie pacienta**

1. Pridanie pacienta do stromu pacientov: O(log(P)), P=počet pacientov
2. Celková zložitosť: O(log(P))

* **Vytvorenie podkladov pre účtovné oddelenie**

1. Nájdenie nemocnice: O(log(N)), N=počet nemocníc
2. Prechádzanie hospitalizácií v danej nemocnici:
   1. Ak je hľadaný mesiac na začiatku: O(l), l=počet prvkov v intervale
   2. Ak je hľadaný mesiac na konci: O(HN), HN=počet hospitalizácií nemocnice N
3. Výpis údajov: O(p\*m), p=počet poisťovní, m=počet dní v mesiaci
4. Celková zložitosť: O(log(N) + HN/2 + p\*m)

* **Výpis aktuálne hospitalizovaných pacientov v nemocnici:**

1. Nájdenie nemocnice: O(log(N)), N=počet nemocníc
2. Prechádzanie všetkými hospitalizovanými pacientami: O(PaN), PaN=počet aktuálne hospitalizovaných pacientov v nemocnici N
3. Celková zložitosť: O(log(N) + log(PaN))

* **Výpis aktuálne hospitalizovaných pacientov, ktorí sú poistencami zadanej poisťovne** **– podľa mena:**

1. Nájdenie nemocnice: O(log(N)), N=počet nemocníc
2. Nájdenie intervalu v strome pacientov zoradených podľa poisťovne a mena: O(log(PaN)+l), PaN=počet aktuálne hospitalizovaných pacientov, l=dĺžka intervalu
3. Celková zložitosť: O(log(N) + log(PaN) + l)

* **Výpis aktuálne hospitalizovaných pacientov, ktorí sú poistencami zadanej poisťovne** – podľa rodného čísla:

1. Nájdenie nemocnice: O(log(N)), N=počet nemocníc
2. Nájdenie intervalu v strome pacientov zoradených podľa poisťovne a rodného čísla: O(log(PaN)+l), PaN=počet aktuálne hospitalizovaných pacientov, l=dĺžka intervalu
3. Celková zložitosť: O(log(N) + log(PaN) + l)

* **Optimalizácia uloženia dát:**

1. Balansovanie všetkých stromových štruktúr v aplikácii: O(SUM(log(ni­)\*ni) cez I), ni=počet prvkov v strome i , I – stromy v aplikácii.
2. Celková zložitosť: O(SUM(log(ni­)\*ni) cez I)

* **Pridanie nemocnice**

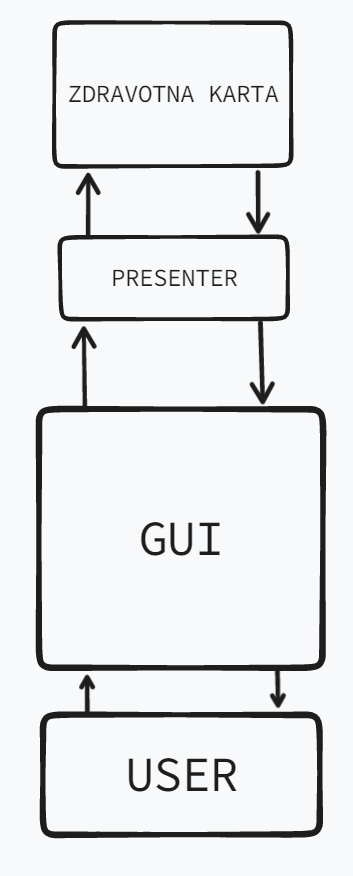
1. Pridanie nemocnice do stromu: O(log(N)), N=počet nemocníc
2. Celková zložitosť: O(log(N))

* **Výpis nemocníc podľa názvu**

1. Prechádzanie všetkých nemocníc: O(N), N=počet nemocníc
2. Celková zložitosť: O(N)

* **Zrušenie nemocnice:**

1. Nájdenie nemocnice, ktorá sa ruší: O(log(N)), N=počet nemocníc
2. Nájdenie nemocnice, ktorá prevezme pacientov: O(log(N)), N=počet nemocníc
3. Import dát z rušenej nemocnice do náhradnej: O(SUM(ni) cez I), ni-počet prvkov v strome i, I=stromy nemocnice a každého aktuálne hospitalizovaného pacienta
4. Aktualizovanie nemocnice v každej hospitalizácii: O(HN), HN=počet hospitalizácií v nemocnici N
5. Zrušenie nemocnice: O(log(N)), N=počet nemocníc
6. Celková zložitosť: O(3log(N) + SUM(ni) cez I + O(HN))
   1. **Grafické rozhranie**

Na obrázku je možné vidieť návrh používateľského rozhrania. Používateľ komunikuje prostredníctvom grafického okna, ktoré udržuje inštanciu presentera. Prostredníctvom objektu presentera sa následne dopytuje aplikácie zdravotnej karty na údaje, ktoré používateľovi zobrazí.

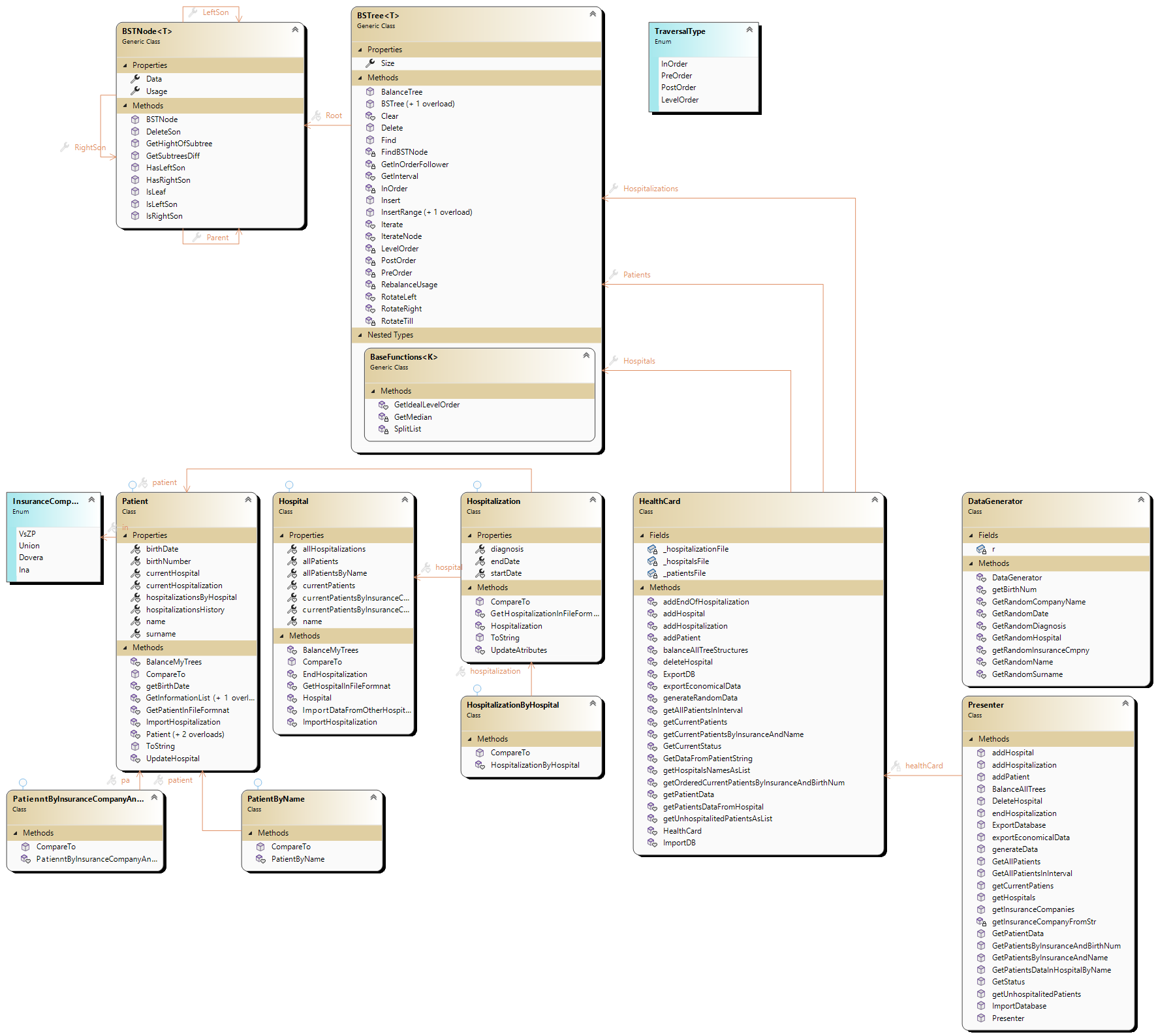
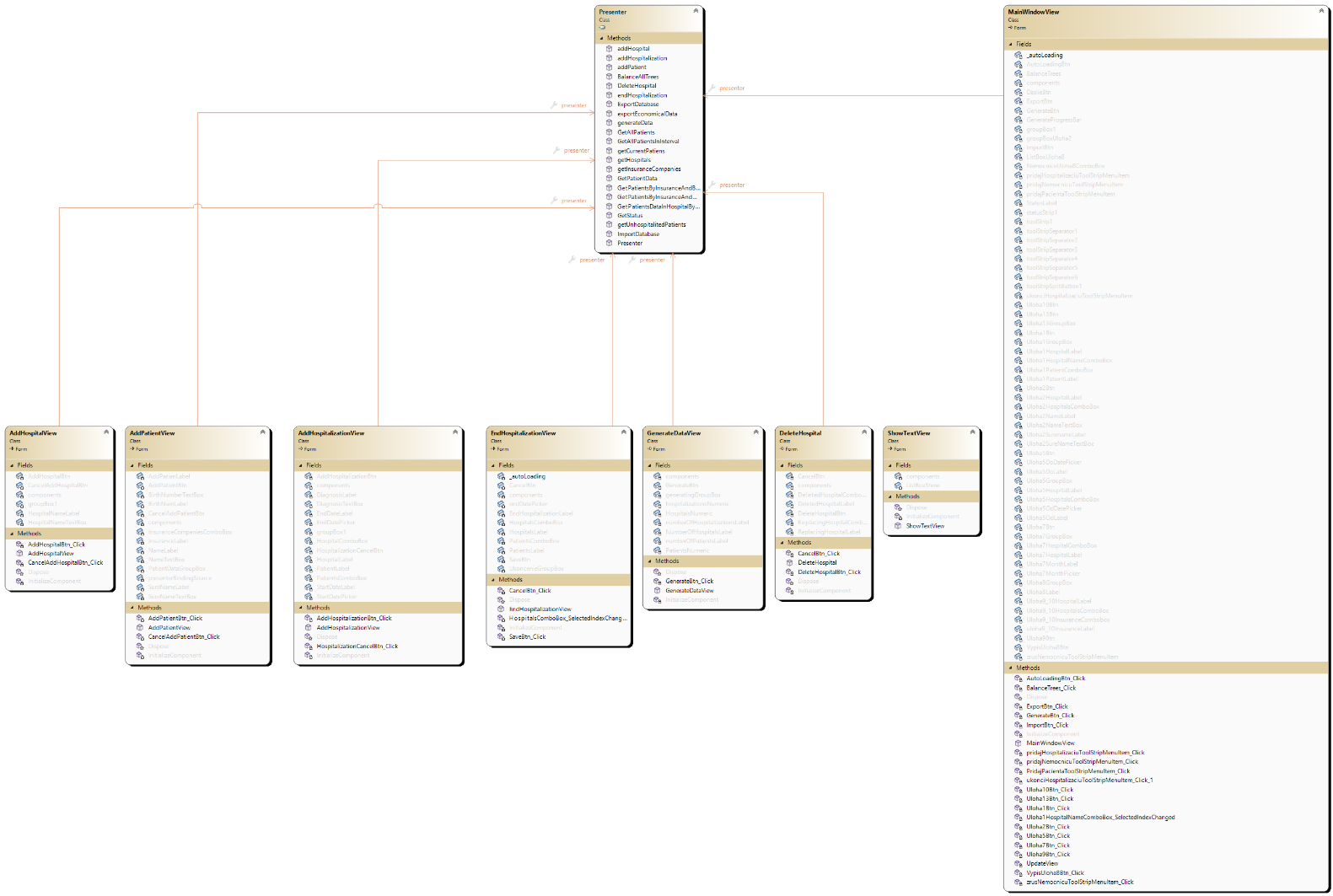
Celá práca je rozdelená na dva projekty, jeden pre aplikáciu a druhý pre grafické rozhranie. V projekte GUI, v triede MainWindowView, sa vytvorí inštancia triedy Presenter, ktorým projekt komunikuje s aplikáciou HealthCard, ktorá je uzavretá v druhom projekte AUS2\_MurinMichal.

* 1. **Import a Export dát**

Pri exporte dát do súboru sa vytvoria tri .csv súbory, každý pre jeden typ dát. Do každého súboru sa postupne uložia základné údaje každého objektu z databázy. To znamená, pre pacienta jeho meno, priezvisko, rodné číslo, pre nemocnicu jej názov. Pre hospitalizáciu ukladáme okrem základných údajov aj názov nemocnice a rodné číslo pacienta. Každý atribút je oddelený bodkočiarkou a každý nový záznam začína na novom riadku

Pri Importe dát sa najskôr importuje súbor s pacientami a nemocnicami, kde sa na základe dát v súbore vytvoria príslušné objekty a uložia sa ako dávka do stromov. Pri importe hospitalizácií, sa podľa názvu nemocnice v súbore nájde už importovaná nemocnica v databáze a priradí sa jej príslušná hospitalizácia. Obdobne to je aj pri pacientovi. Po úspešnej aktualizácii pod-stromov v pacientovi aj nemocnici sa hospitalizácie vložia v dávke do databázy.

# **Prílohy:**

1. **UML Diagram Projektu AUS2\_MurinMichal**
2. **UML Diagram projektu GUI:**