**ŽILINSKÁ UNIVERZITA V ŽILINE**

FAKULTA RIADENIA A INFORMATIKY

Algoritmy a údajové štruktúry 2

MATEJ MAŽGÚT

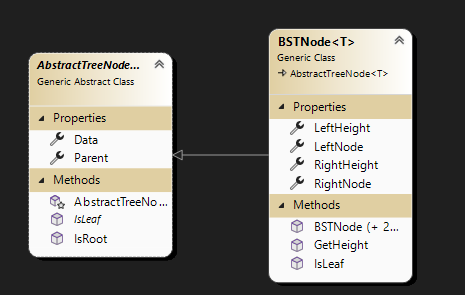
**Elektronická zdravotná karta**

# Použité údajové štruktúry

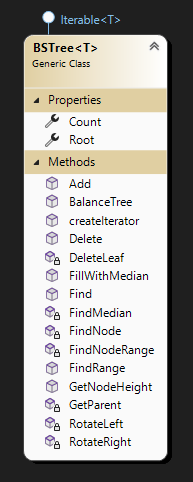
V projekte budem využívať mnou implementovanú štruktúru binárneho vyhľadávacieho stromu. V tejto štruktúre som tiež implementoval vyvažovací algoritmus, ktorý pomocou rotácii preusporiada prvky štruktúry a vznikne tak vyvážený strom.

## Metóda vyváženia stromu

Okrem odkazu na otca, ľavého a pravého syna si ešte v každom prvku stromu ukladám aj výšku ľavého a pravého podsromu prvku.

Trieda prvoku stromu teda vyzerá nasledovne:

Prvok obsahuje ešte metódu GetHeight(). Táto metóda vráti rozdeľ výšok pravého a ľavého podstromu. Neskôr túto hodnotu využívam pri vyvažovacom algoritmy v samotnom strome. Popis jeho triedy je na obrázku vpravo.

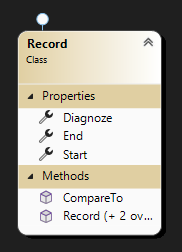
Strom obsahuje metódy RotateRight() a RotateLeft(), ktoré zrotujú vrhol stromu a zmenia výšky ostatných vrcholov. Následne sa tu nachádza metóda BalanceTree(), ktorá strom pomocou rotácií vybalancuje a vznikne tak vyvážený strom, kde rozdiel výšok ľavého a pravého podstromu je maximálne jeden.

V metóde uplatňujem nasledovný algoritmus:

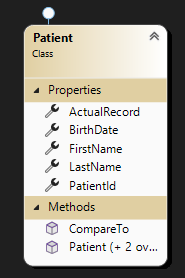
1. Vytvorím level-order prehliadku stromu a uložím si ju do štruktúry Stack
2. Vyberiem prvok zo Stacku a nastavím mu lávu a pravú výšku. Nastavím ju tak, že sa pozriem na maximálnu výšku potomkov vrcholu a pripočítam k ním číslo jeden.
3. Zavolám metódu GetHeight(). Ak metóda vráti číslo väčšie ako jeden alebo menšia ako mínus jeden rotujem vrchol až kým rozdiel jeho výšok v absolútnej hodnote nie je menší ako dva.
4. Ak Stack nie je prázdny vraciam sa na krok 2.
5. Skontrolujem výšky všetkých vrcholov, ak narazím na vrchol s väčšou výškou ako 1 alebo menšou ako -1 vraciam sa na krok 1

## Návrh údajových štruktúr a tried

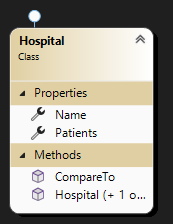
Ukladané dáta som si rozdelil do štyroch hlavných tried:

1. Record viď. obrázok

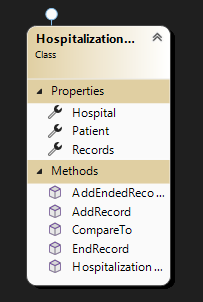
Táto trieda predstavuje záznam o hospitalizácií pacienta. Obsahuje začiatočný dátum hospitalizácie, koncový dátum hospitalizácie a popis diagnózy pacienta.

1. Patient viď. obrázok

Trieda Patient bude obsahovať základné informácie o pacientovi a tiež odkaz na záznam hospitalizácie, ak je pacient aktuálne nikde hospitalizovaný.

1. Hospital viď. obrázok

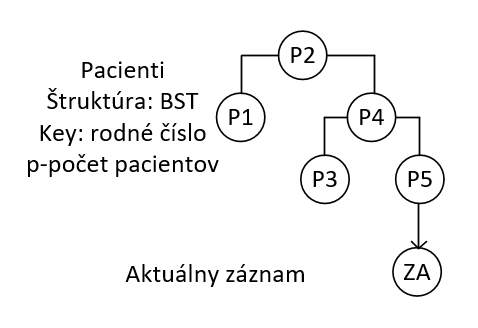
Trieda opisuje nemocnicu a tiež všetkých aktuálne hospitalizovaných pacientov.

1. HospitalizationRecord viď. orázok

HospitalizationRecord bude slúžiť na uchovávanie všetkých záznamoch pacienta v konkrétnej nemocnici.

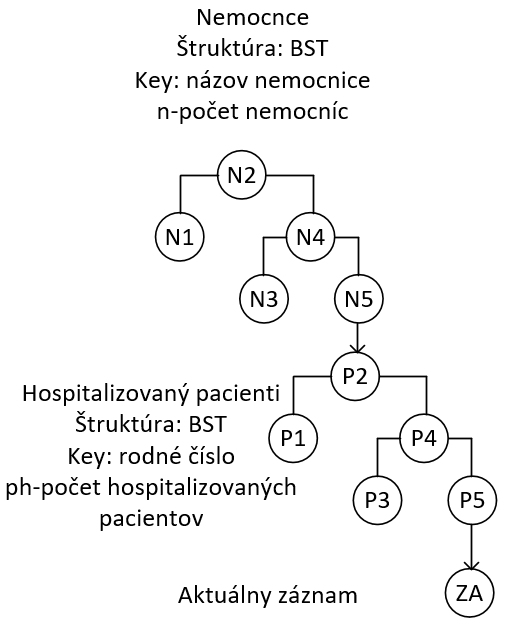
Popísané triedy si ukladám to troch hlavných binárnych stromov:

1. Binárny strom pacientov



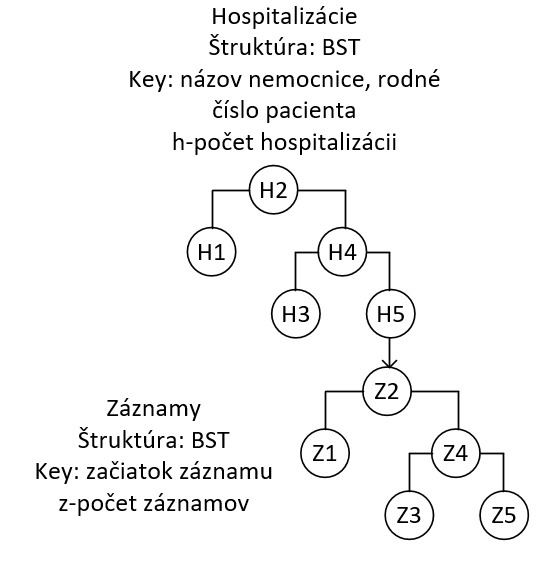
V štruktúre sa nachádzajú všetci existujúci pacienti. V prípade pridania nového pacienta, hladina pacienta alebo jeho vymazania vykonávam tieto operácie so zložitosťou log2(p).

1. Binárny strom nemocníc



Štruktúra predstavuje zoznam všetkých dostupných nemocníc. Pridanie novej nemocnice, nájdenie nemocnice a vymazanie nemocnice ma zložitosť log2(n). V každej nemocnici sa nachádza ešte binárny strom s aktuálne hospitalizovanými pacientami. Ak chcem teda do nemocnice pridať hospitalizovaného pacienta, nájsť ho alebo ho vymazať, vykonávam to so zložitosťou log2(n) + log2(ph).

1. Binárny strom hospitalizácií



Posledný binárny strom predstavuje záznamy o všetkých hospitalizáciách. Každá hospitalizácia sa skladá z kompozitného kľúča, ktorý je tvorený názvom nemocnice a rodným číslom pacienta. Počet prvkov v tejto štruktúre a teda h(počet hospitalizácii) je n(počet nemocníc)\*p(počet pacientov). Ak budem teda pridávať novú hospitalizáciu, hľadať hospitalizáciu alebo ju odstraňovať, vykonávam to so zložitosťou log2(h) = log2(n) \* log2(p). Každá hospitalizácia ešte obsahuje binárny strom všetkých záznamov pacienta v danej nemocnici. Vyhľadávanie, mazanie alebo pridanie záznamu vykonávam so zložitosťou log2(z), kde z predstavuje počet záznamov pacienta v konkrétnej nemocnici.