**FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ SLOVENSKÁ TECHNICKÁ UNIVERZITA**

lkovičova 2, 842 16 Bratislava 4

2022/2023

Dátové štruktúry a algoritmy

Zadanie č.1

# Úvod

Účelom tohto článku je zdokumentovať moju prácu na 1. zadaní v predmete Dátové štruktúry a algoritmy. Zdrojový kód som písal v jazyku C++. V prvom semestri som sa naučil jazyk C, no C++ má napríklad knižnicu list, typ auto a niekoľko ďalších vecí, ktoré jazyku C chýbajú a pri vypracovávaní tohto zadania mi zjednodušili život, a navyše som sa naučil nový programovací jazyk.

Štruktúra tejto dokumentácie je nasledovná: v prvej sekcii je popísaný zdrojový kód pre implementáciu dátových štruktúr. V druhej sekcii sa venujem testovaniu časovej náročnosti mojich algoritmov a na záver rozoberám výsledky testovania.

# Zdrojové kódy

## AVL Strom

Ako prvú dátovú štruktúru som si vybral AVL strom. Definoval som si štruktúru Node (uzol stromu) v ktorej sa uchovávajú vstupné dáta. Aby som sa naučil pracovať s objektovou paradigmou, ktorá bola pre mňa nová, implementoval som ho v triede – AVLTree, kde som jednotlivé funkcie zadefinoval ako metódy tejto triedy.

Text

Description automatically generated

V hlavičke sú zahrnuté knižnice:

* <iostream> na prácu s terminálom – input-output
* <algorithm> na rôzne funkcie, použil som iba max()
* <fstream> na prácu s textovými súbormi – tie som použil na testovanie
* <string> na jednoduchšiu prácu s reťazcami
* <chrono> na účely merania času

Štruktúra uzla obsahuje:

* Int key – uchováva číselnú hodnotu uzla
* Int height – uchováva výšku uzla v strome (súčasť algoritmu na vyvažovanie)
* String name – uchováva extra informácie asociované s číselnou hodnotou key
* Ukazovatele na dieťa s menšou hodnotou a dieťa s väčšou hodnotou

V triede som ďalej definoval konštruktor **AVLTree()** , ktorý nastaví hodnotu pomocného pointera root (koreň) na NULL a deštruktor **~AVLTree()**, ktorý volá funkciu **destructTree(\*\*node)**. Deštrukciu stromu som vyriešil takýmto spôsobom preto, lebo najjednoduchšie sa to celé premaže rekurzívne, no deštruktor (ani konštruktor) by nemal mať rekurziu.

### Balansovanie

Na účely vyvažovania stromu som si zadefinoval pomocné funkcie **getHeight(\*node)**, **getBalance(\*node)** a **updateHeight(\*node)**, tieto sú ďalej využité vo funkcii **balanceNode(\*node)**.

Funkcia **getHeight(\*node)** je veľmi jednoduchá, no veľa krát sa používa a preto je jednoduchšie si ju zadefinovať zvlášť. Ak uzol neexistuje (\*node == NULL), vráti 0, inak vráti jeho výšku. Týmto spôsobom sa vyhýbam možnému segmentation faultu.

Funkcia **getBalance(\*node)** vráti 0, ak uzol neexistuje (\*node == NULL), inak vypočíta a vráti balans daného uzla: getHeight(ľavé dieťa) - getHeight(pravé dieťa).

Funkcia **updateHeight(\*node)** sa volá po vložení alebo vymazaní uzla a slúži na aktualizáciu výšky daného uzla. Funkcia **max(...)** vráti maximálnu hodnotu z výšok detí a k tomu sa pripočíta +1.

Text

Description automatically generatedText

Description automatically generated

Vyššie spomenuté funkcie majú ako som už spomenul uplatnenie vo funkcii balanceNode(\*node). V tejto funkcii sa na základe hodnoty balansu (int bf) volajú jednotlivé rotácie:

* Ak je balans uzla > 1 a balans jeho menšieho dieťaťa je < 0,

zavolá sa najprv **leftrotate(\*node)** a potom **rightRotate(\*node)**

* Ak je balans uzla > 1 a balans jeho menšieho diťaťa je >= 0,

zavolá sa iba **rightRotate(\*node)**

* Ak je balans uzla < -1 a balans jeho väčšieho dieťaťa je > 0,

zavolá sa najprv **rightRotate(\*node)** a potom **leftRotate(\*node)**

* Ak je balans uzla < -1 a balans jeho väčšieho dieťaťa je <= 0,

zavolá sa iba **leftRotate(\*node)**

Text

Description automatically generatedTieto rotácie sú jadrom balansovania pomocou AVL algoritmu, a zabezpečujú fakt, že v AVL strome nikdy nie je niektorá vetva dlhšia (vyššia) o viac ako jeden uzol od druhej. Vďaka tomu sú AVL stromy kompletne vybalansované, narozdiel od mnohých iných stromov, ktoré sú iba približne vybalansované.

Funkcia **leftRotate(\*node)** dostáva ukazovateľ na uzol (node), podľa ktorého chceme strom otočiť doľava. Do pomocnej premennej (temp) uloží väčšie dieťa tohto uzla, a do ďalšej pomocnej premennej (temp2) uloží menšie dieťa štruktúry uloženej v temp. Tieto kroky sú nevyhnutné na to, aby sa zabránilo strácaniu detí alebo porušeniu vlastností stromu. Ďalej sa zamenia temp za pôvodný uzol, aktualizuje sa výška uzlov, ktoré sa vymieňali a vracia sa ukazovateľ temp.

Funkcia **rightRotate(\*node)** robí analogický opak funkcie ľavého otáčania. Praktický príklad týchto rotácii je uvedený na obrázkoch nižšie.

Diagram

Description automatically generated

Diagram

Description automatically generated