SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

# FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Binárne rozhodovacie diagramy

ZADANIE 2

**Mária Matušisková AID ID: 116248**

**DSA**

**Cvičiaci: Ing. Martin Komák, PhD. (Po 16:00) 2023**

**Binárne rozhodovacie diagramy**

**ZADANIE 2**

**Študijný program: Informatika, konverzný**

**Študijný odbor: 9.2.1 Informatika**

**Školiace pracovisko: Fakulta informatiky a informačných technológií**

### **2023 Mária Matušisková**

**Obrázok, na ktorom je text, snímka obrazovky, písmo, papier

Automaticky generovaný popis**

**Obrázok, na ktorom je text, papier, písmo, dokument

Automaticky generovaný popis**

**Obrázok, na ktorom je text, písmo, snímka obrazovky, papier

Automaticky generovaný popis**

Obsah

[FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ 1](#_Toc134582857)

[**2023** **Mária Matušisková** 2](#_Toc134582858)

[1 Úvod projektu 6](#_Toc134582859)

[2 Spracovanie vstupu 7](#_Toc134582860)

[3 Hashovacie tabuľky 9](#_Toc134582861)

[4 Uzly (Nodes) 11](#_Toc134582862)

[**4.1** **trieda Node** 11](#_Toc134582863)

[**4.2** **trieda LeafNode** 12](#_Toc134582864)

[5 Binárne rozhodovacie diagramy 13](#_Toc134582865)

[**5.1** **ROBDD** 13](#_Toc134582866)

[**5.2.1** **BDD\_create** 14](#_Toc134582867)

[**5.2.2** **BDD\_create\_with\_best\_order** 15](#_Toc134582868)

[**5.2.3** **BDD\_use** 17](#_Toc134582869)

[6 Testovanie 17](#_Toc134582870)

[**6.1** **Generovanie náhodného boolean výrazu s určitým množstvom premenných** 18](#_Toc134582871)

[**6.2** **Generovanie náhodných binárnych kombinácii s určitým množstvom premenných** 20](#_Toc134582872)

[**6.3** **Metóda, ktorá vyhodnocuje pravdivostnú tabuľku pre jediný prípad** 21](#_Toc134582873)

[**6.4** **Main** 22](#_Toc134582874)

[**6.5** **Časová a priestorová zložitosť a percentuálna miera redukcie** 23](#_Toc134582875)

[7 Záver 25](#_Toc134582876)

[Zdroje 26](#_Toc134582877)

# **Úvod projektu**

Projekt bol realizovaný vo vývojárskom prostredí IntelliJ IDEA v programovacom jazyku Java na počítači MacBook Pro (13-inch, M1, 2020). Boli použité niektoré me- tódy z knižnice *java.util.\*.* Osnova projektu je rozdelená do pod balíkov v hlavnej doméne v reverznom tvare *sk.stuba.fiit.\*.*

Obrázok, na ktorom je text, snímka obrazovky, písmo

Automaticky generovaný popis

Obrázok 1, Ukážka rozloženia balíkov v projekte

Na [obrázku 1](#Obrazok1) je ukázané ako sú jednotlivé súbory rozdelené do štyroch časti. Obsahom vrchného balíka sú **logické operátory**, ktoré slúžia na spracovanie vstupu.

Nadchádzajúci balík obsahuje hashovacie tabuľky. Tabuľka **StoreTableBDD** ukladá všetky vytvárajúce sa uzly, vrátane 0 a 1. Druhá tabuľka **ReductionTableBDD** je pomocná a slúži na kontrolu pri redukovaní.

Tretí balík obsahuje samotný rozhodovací diagram spolu s jeho potomkami, teda uzlami v strome a listami.

Posledná zložka je samotný **program**, kde testujeme funkčnosť diagramu.

# **Spracovanie vstupu**

Využívajú sa dve [abstraktné triedy](#Obrazok2), ktoré dedia logické spojky. V BDD sa vo funkcii BDD\_create zavolá konštruktor Or a ten automaticky zavolá aj ostatné logické operátory a tak sa postupne [parsuje](#Obrazok3) boolean funkcia. Každá spojka má vlastný list children, kde má napísané svoje hodnoty. Napríklad list s názvom children spojky And by vyzeral takto: {“A.B“, “C.D“, ...}.

Toto rozdelenie sa potom používa pri ďalších funkciách [evaluate](#Obrazok3) a [replace,](#Obrazok4) ktoré slúžia na ohodnotenie výrazu, niečo ako pravdivostná tabuľka.

Obrázok, na ktorom je text, snímka obrazovky, softvér, počítač

Automaticky generovaný popisObrázok, na ktorom je text, snímka obrazovky, softvér, písmo

Automaticky generovaný popis

Obrázok 2, abstraktné triedy pre spracovanie výrazu

Obrázok, na ktorom je text, snímka obrazovky

Automaticky generovaný popis

Obrázok 3, konštruktory triedy Or a funkcia parseInput

Obrázok, na ktorom je text, snímka obrazovky

Automaticky generovaný popis

Obrázok 4, funkcia evaluate()

Obrázok, na ktorom je text, snímka obrazovky, písmo

Automaticky generovaný popis

Obrázok 5, funkcia replace(), slúži pri evaluacii uzlov 0 alebo 1

# **Hashovacie tabuľky**

Slúžia na kontrolu pri redukovaní a vytváraní uzlov. V tomto projekte sa používajú dve tabuľky:

* [StoreNodeBDD](#Obrazok6)
* [ReductionTableBDD](#Obrazok7)

Obrázok, na ktorom je text, snímka obrazovky, softvér

Automaticky generovaný popis

Obrázok 6, StoreNodeBDD, tabuľka pre ukladanie uzlov diagramu

StoreNodeBDD ukladá jednotlivé rozložené booleovské výrazy do tabuľky. Ukladajú sa iba tie zredukované. Tabuľka nie je nikdy prázdna, prvé dve pozície má vždy vyhradené pre posledné listy 0 a 1. Tak sa overí, že diagram bude obsahovať vždy iba dva listy.

Obrázok, na ktorom je text, snímka obrazovky, softvér, multimediálny softvér

Automaticky generovaný popis

Obrázok 7, ReductionTableBDD, tabuľka určená na kontrolovanie redukcie

Tabuľka určená pre kontrolu redukcie. Keď sa v diagrame vytvára nový uzol, tak jednou z podmienok je, či sa už daný uzol s hodnotami nenachádza v strome, ak áno vráti ten.

# **Uzly (Nodes)**

## **4.1 trieda Node**

Pri ich vytváraní sa do konštruktora posiela jeho index, ľavý a pravý potomok a výraz. Tieto hodnoty sú potrebné pri redukcii.

Obrázok, na ktorom je text, snímka obrazovky, písmo

Automaticky generovaný popis

Obrázok 8, konštruktor triedy Node

Koreň diagramu ma trochu odlišnú stavbu, rozširuje sa ešte o premenné **order** a **reductionPercentage**. Tieto premenné sú potrebné hlavne pre testovacie účely.

Pri kontrole redukcie sa volá aj prekonaná metóda [equals(),](#Obrazok9) ktorá porovnáva pravého s ľavým potomkom.

Obrázok, na ktorom je text, snímka obrazovky, písmo

Automaticky generovaný popis

Obrázok 9, metóda equals() v triede Node

## **4.2 trieda LeafNode**

Táto trieda je určená iba pre výsledné výrazy. Podľa vyhodnotenia obsahuje hodnotu true alebo false.

Obrázok, na ktorom je text, snímka obrazovky, softvér

Automaticky generovaný popis

Obrázok 10, trieda LeafNode s metódou equals()

# **Binárne rozhodovacie diagramy**

Inak povedané aj binárny rozhodovací strom, je to dátová štruktúra, ktorá rozhoduje prechodom od začiatku stromu, teda koreňa až po výsledok. Každý uzol obsahuje jedno z rozhodnutí záležiac od vstupu 0 alebo 1.

Tento projekt je zameraný na Booleovskú funkciu **DNF** (disjunktívna normálová forma, súčet súčinov). Taktiež sa využíva Shannonova dekompozícia (prechádzanie zhora nadol), ktorá je užitočná najmä vtedy, keď sa vo vstupe nevyskytuje premenná vo všetkých podvýrazoch a tak sa použije pre obidve strany diagramu.

## **ROBDD**

**ROBDD (Reduced Ordered Binary Decision Diagram)** využíva tri hlavne funkcie určené podľa zadania:

* Node **BDD\_create(**String bfunction, String order)
* Node **BDD\_create\_with\_best\_order**(String bfunction)
* String **BDD\_use(**Node node, String inputs)

Do [konštruktora](#Obrazok11) sa posiela boolean funkcia a následne sa automaticky vypočíta počet premenných a zadefinujú sa hashovacie tabuľky.

Obrázok, na ktorom je text, snímka obrazovky, písmo

Automaticky generovaný popis

Obrázok, na ktorom je text, snímka obrazovky, písmo

Automaticky generovaný popis

Obrázok 11, konštruktor ROBBD

### **5.2.1 BDD\_create**

Funkcia je rozdelená do dvoch časti, [BDD\_create a BDD\_create\_helper.](#Obrazok12)

V prvej metóde sa parsuje vstup a taktiež sa ukladá postupnosť premenných, keďže hľadáme tú najlepšiu kombináciu stromu. Návratový hodnota je typu Node a konkrétne sa vracia koreň stromu.

V BDD\_create\_helper už ide o samotné generovanie binárneho rozhodovacieho stromu. Ide o rekurzívny typ funkcie, ktorá sa volá zakaždým keď sa má vytvoriť nový uzol. Ten sa vytvorí len vtedy keď sú splnené všetky podmienky, tie overujeme v ďalšej metóde checkReduction().

Obrázok, na ktorom je text, snímka obrazovky, písmo

Automaticky generovaný popis

Obrázok, na ktorom je text, snímka obrazovky, softvér

Automaticky generovaný popis

Obrázok 12, metódy BDD\_create a BDD\_create\_helper

[Metóda](#Obrazok13) určená na redukciu kontroluje či výrazy sú súmerné a či sa už taká hodnota nachádza v tabuľke. Pri týchto podmienkach sa taktiež inkrementuje premenná beforeReduction, tak sa spočítavajú jednotlivé uzly v strome, ktoré sa zredukovali.

Počítanie po redukcií nie je potrebné, pretože číslo sa vie získať z tabuľky StoreNodeBDD, volaním funkcie getSize().

Obrázok, na ktorom je text, snímka obrazovky, písmo

Automaticky generovaný popis

Obrázok 13, metóda checkReduction()

Časová zložitosť BDD\_create, BDD\_create helper a checkReductionBDD je konštantná, teda **O(1).**

### **5.2.2 BDD\_create\_with\_best\_order**

Najlepšie vyhodnotený diagram sa hľadá lineárnym poradím premenných (rotovanie), ktoré sa posielajú do metódy BDD\_create a výsledok porovnávame s ostatnými. Návratový typ je Node a presnejšie ide o root najlepšieho nájdeného stromu.

Počas merania sa kalkuluje aj percentuálna miera zredukovania.

Časová zložitosť zaberie dokopy **O(n),** avšak záleží aj od počtu premenných.

Obrázok, na ktorom je text, snímka obrazovky

Automaticky generovaný popis

Obrázok, na ktorom je text, snímka obrazovky, multimediálny softvér

Automaticky generovaný popis

Obrázok 14, metóda BDD\_create\_with\_best\_order

### **5.2.3 BDD\_use**

[BDD\_use](#Obrazok15) vyhodnocuje samotný diagram pomocou reťazového vstupu v binárnej sústave dĺžky zoznamu premenných vo výraze.

Ak je výsledok rovný false, vráti sa 0, ak true tak 1. Ak je výraz neplatný, vracia sa -1 ako neúspešný pokus.

Obrázok, na ktorom je text, snímka obrazovky, softvér, písmo

Automaticky generovaný popis

Obrázok 15, metóda BDD\_use

Časová náročnosť je **O(n)** závisí od dĺžky reťazca, teda od počtu premenných.

# **Testovanie**

Testovalo sa volanie funkcii BDD\_create, BDD\_create\_with\_best\_order a BDD\_use so 100 iteráciami. V programe sú tri testovacie scenáre a počtom premenných:

* 3
* 13
* 20

Pri metóde BDD\_use sa generovala ešte aj funkcia TruthTable pre overenie očakávaného výsledku.

Počas iterovania programu sa pri každom volaní vypíše, koľko uzlov bolo zredukovaných a koľko by ich bolo bez nej. Taktiež sa vypočíta percentuálna miera jednotlivo, ale aj spriemerovane.

Výsledok sa väčšinou hýbal okolo 99%. Okrem toho program kalkuluje aj dobu trvanie každej z troch funkcii a priemer je napísaný v nanosekundách.

## **Generovanie náhodného boolean výrazu s určitým množstvom premenných**

Obrázok, na ktorom je text, snímka obrazovky, softvér

Automaticky generovaný popisPremenné sú náhodne generované podľa latinskej abecedy. Šanca, že premenná je negovaná je 50%. Okrem premenných sa náhodne hľadá aj počet logických operandov a kombinácii.

Obrázok 16, metóda generateBooleanFunction()

Obrázok, na ktorom je text, snímka obrazovky, písmo

Automaticky generovaný popis

Obrázok 17, metódy generateVariables() a generateClauses()

Obrázok, na ktorom je text, snímka obrazovky, písmo

Automaticky generovaný popis

Obrázok 18, metóda generateRandomCombination()

Obrázok, na ktorom je text, snímka obrazovky, písmo

Automaticky generovaný popis

Obrázok 19, generateMissingVariables()

## **Generovanie náhodných binárnych kombinácii s určitým množstvom premenných**

Podľa počtu premenných na vstupe sa vygeneruje náhodná kombinácia premenných v binárnej sústave.

Obrázok, na ktorom je text, snímka obrazovky, písmo

Automaticky generovaný popis

Obrázok 20, metóda generateBinaryCombinations()

## **Metóda, ktorá vyhodnocuje pravdivostnú tabuľku pre jediný prípad**

Táto funkcia testuje očakávaným výsledok s výsledkom z binárneho rozhodovacieho diagramu. Pri tomto vyhodnocovaní sa využíva balík MathLogic, o ktorom bolo už písané v 2.kapitole.

Obrázok, na ktorom je text, snímka obrazovky

Automaticky generovaný popis

Obrázok 21, metóda TruthTable()

## **Main**

Samotné testovanie a využívanie funkcii s výpismi.

Test na 3 a 13 premenných.

Obrázok, na ktorom je text, snímka obrazovky, softvér, písmo

Automaticky generovaný popis

Obrázok, na ktorom je text, snímka obrazovky

Automaticky generovaný popis

Obrázok 22, metóda main()

Podobne vyzerá test s prípadom 20 premenných.

## **Časová a priestorová zložitosť a percentuálna miera redukcie**

|  |  |  |  |
| --- | --- | --- | --- |
| **BDD\_create, BDD\_create\_with\_best\_order, BDD\_use** | | | |
| **n** | **create (ns)** | **create\_with\_best\_order (ns)** | **use (ns)** |
| 0 | 0 | 0 | 0 |
| 2 | 359007 | 514767 | 767 |
| 4 | 58362 | 741341 | 1057 |
| 6 | 98533 | 1387148 | 1496 |
| 8 | 347208 | 4482177 | 3074 |
| 10 | 833260 | 11068589 | 5209 |
| 12 | 2161685 | 30264796 | 4482 |
| 14 | 8325329 | 138519410 | 6757 |
| 16 | 36119010 | 639652350 | 7232 |
| 18 | 139325101 | 2699963489 | 8039 |
| 20 | 596112185 | 12573038005 | 12388 |

|  |  |
| --- | --- |
| **Percentuálna miera redukovania** | |
| **n** | **%** |
| 0 | 0 |
| 2 | 43,22 |
| 4 | 43,66 |
| 6 | 82,99 |
| 8 | 93,98 |
| 10 | 97,77 |
| 12 | 99,21 |
| 14 | 99,69 |
| 16 | 99,89 |
| 18 | 99,96 |
| 20 | 99,98 |

# **Záver**

Po preskúmaní vybraných algoritmov sa zistilo, ktoré štruktúry sú vhodné na rôzne metódy. Podľa toho na čo sa chce program najviac orientovať.

Pri stromoch. Je celkovo najlepší AVL strom, následne červeno-čierny strom a ako posledný nevyvážený binárny vyhľadávací strom. Z grafu bolo vidieť, že AVL strom narastal čo najviac súvisle, kolízie vznikali pri BVS. Celkovo najpomalší bol červeno-čierny strom, avšak pri testovaní jednotlivých metód, mal časy pomerne dobré.

V hašovacích tabuľkách sa ako efektívnejšie riešenie kolízii prejavilo otvorené adresovanie. Reťazenie vynikalo iba pri funkcii *insert().*

Projekt bol realizovaný v jazyku Java, je možné, že v iných jazykoch ako napríklad C sa môže časová efektivita meniť. Pretože knižnice tam môžu byť implementované inak.

# **Zdroje**

**Binárny vyhľadávací diagram**

* [**https://www3.cs.stonybrook.edu/~cram/cse505/Fall20/Resources/bdd97.pdf**](https://www3.cs.stonybrook.edu/~cram/cse505/Fall20/Resources/bdd97.pdf)
* Adam Gábor, AIS ID 116174, DSA documentation