**Dátové štruktúry a algoritmy**

**Zadanie č. 2**

Obsah

[**1.** **Binárne rozhodovacie diagramy** 2](#_Toc134474763)

[**1.1** **Fungovanie binárneho rozhodovacieho diagramu** 2](#_Toc134474764)

[**1.2** **Vytvorenie BDD** 2](#_Toc134474765)

[**BDD\_create:** 2](#_Toc134474766)

[**Decompose:** 4](#_Toc134474767)

[**Reduce:** 4](#_Toc134474768)

[**1.3** **Nájdenie najvýhodnejšieho BDD** 4](#_Toc134474769)

[**BDD\_create\_with\_best\_order:** 4](#_Toc134474770)

[**getUniqueCharacters:** 5](#_Toc134474771)

[**1.4** **BDD\_use** 6](#_Toc134474772)

[**2.** **Testovanie** 8](#_Toc134474773)

[**2.1** **Priebeh testovania** 8](#_Toc134474774)

[**2.2** **Výsledky testovania** 8](#_Toc134474775)

[**3. Výsledok** 10](#_Toc134474776)

# **Binárne rozhodovacie diagramy**

## **Fungovanie binárneho rozhodovacieho diagramu**

Binárny rozhodovací diagram (BDD) je dátovou štruktúrou, ktorá obsahuje root v ktorom je uložená vstupná formula. Na každý uzol napájajú 2 child nody. Low child pre hodnotu 0 a high child pre hodnotu 1. V tomto zadaní budeme reprezentovať Booleovské funkcie. Podľa jednotlivých premenných sa strom rozkladá až kým sa nevytvorí list s jeho poslednou premennou alebo negovaným tvarom.

Obrázok, na ktorom je diagram

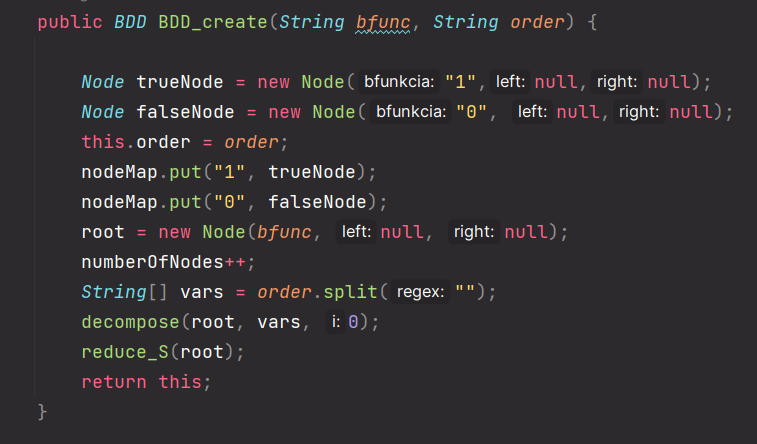
Automaticky generovaný popis

Obrázok: Vizualizácia rozloženia BDD

## **Vytvorenie BDD**

### **BDD\_create:**

Vytvorím 2 nody, ktoré obsahujú iba 1 alebo 0. A vytvorím root node, ktorý pošlem do metódy decompose(), kde sa celá funkcia rozloží rekurzívnym volaním. Po vytvorení stromu sa zavolá metóda reduce\_S, ktorá prejde celý strom a odstráni nody s 2 rovnakými child nodmi.



Obrázok, na ktorom je text

Automaticky generovaný popis

Časť metódy decompose(), kde sa tvoria nové nody.

### **Decompose:**

Volá sa dokým decompose neprejde všetky premenné pre každú vetvu stromu. Na začiatku sa vytvoria dve polia, ktoré reprezentujú low child a high child (v mojej implementácii left a right). Hlavný cyklus prechádza cez jednotlivé klauzuly a rozdeľuje premenné do leftClause a rightClause podľa výskytu. Ak sa v klauzule premenná nenachádza pridáme ju do oboch polí. Ďalej mažeme rovnaké premenné z klauzúl. A po mazaní kontrolujeme či sa celá klauzula nedá vyhodnotiť ako 1 alebo 0. Ak nie tak vytvoríme nový node a voláme decompose(). Ak je 1 alebo 0 tak ju nastavíme na true alebo false node, ktorý sme vytvorili v BDD\_create. Na konci sa node redukuje a pridáva do hash tabuľky

### **Reduce:**

Reduce vykonáva redukciu typu I. To znamená, že ak node s rovnakou funkciou existuje v nodeMap tak sa nastaví na vytvorený a netvorí sa nový, ktorý je rovnaký. Inak sa vytvorený node pridá do nodeMap.Obrázok, na ktorom je text

Automaticky generovaný popis

## **Nájdenie najvýhodnejšieho BDD**

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

Metóda má za úlohu vytvoriť N (v mojom prípade 100) stromov s rôznymi postupnosťami premenných. Ak je novo vytvorený strom menší ako uložený tak sa nahradí novo vytvoreným.

Obrázok, na ktorom je text

Automaticky generovaný popis

### **getUniqueCharacters:**

Metóda slúži na získanie unikátnych charakterov z formuly. Prechádza všetky znaky a kontroluje či sa nachádzajú v poli ak nie pridá ich. Na záver ich zamieša a vytvorí string.

Obrázok, na ktorom je text

Automaticky generovaný popis

## **BDD\_use**

Slúži na overenie či bol daný strom správne vytvorený. Najprv zistí správny výsledok z formuly a uloží jeho hodnotu. Overuje ho tak, že za premenné dosadí 1 a 0 podľa vstupu. Potom cyklom prejdem jednotlivé klauzuly dokým nenájdem jednu bez 0 alebo dokým ich neprejdem všetky. Môj strom získa char podľa ktorého bola formula rozkladaná a zistí jej hodnotu a podľa toho sa posunie vľavo alebo vpravo v strome. Výsledok sa uloží a potom sa oba porovnajú. Ak sa nerovnajú metóda vráti -1 ak sa rovnajú tak vráti 1 alebo 0 podľa pravdivosti.

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok, na ktorom je text

Automaticky generovaný popis

# **Testovanie**

## **Priebeh testovania**

Zavolal som najprv metódu generateBfunctions(), ktorá mi vygenerovala 200 funkcií. V tejto metóde sa dá volí koľko rôznych premenných má formula obsahovať, tiež sa dá zvoliť počet premenných v klauzule aj počet klauzúl. Ďalej prvým cyklom prejdem vygenerované formuly a pre každú vytvoríme BDD s náhodným orderom. Každý vytvorený strom prejdeme aj všetkými možnosťami s BDD\_use pre kontrolu správnosti stromu. Druhým cyklom vytvoríme best order pre všetky formuly z poľa. Taktiež vytvorené stromy otestujeme s BDD\_use.

## **Výsledky testovania**

Tabuľka pre BDD\_create, v ktorej sa nachádzaju priemerné hodnoty z 200 opakovaní pre BDD\_create. Čas je uvedený v mikrosekundách a miera zredukovania v percentách. Do počtu nodov som nerátal node s hodnotou 1 alebo 0.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Počet premenných | Čas | Počet nodov | Počet po zredukovaní | Miera zredukovania |
| 10 | 529,29 | 323,63 | 60,17 | 80,68 |
| 11 | 754,74 | 549,24 | 73,55 | 85,84 |
| 12 | 944,19 | 1014,39 | 90,44 | 90,34 |
| 13 | 1403,26 | 1723,23 | 105,57 | 93,42 |
| 14 | 1787,57 | 3061,21 | 118,96 | 95,67 |
| 15 | 2996,85 | 5777,33 | 143,03 | 97,20 |
| 20 | 34175,35 | 98483,15 | 235,73 | 99,61 |

Tabuľka pre BDD\_create\_with\_best\_order, v ktorej sa ako v predchádzajúcej tabuľke nachádzajú priemerné hodnoty z 200 opakovaní pri tvorení best orderu. Čas vytvorenia je pre najlepší nájdený strom.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Počet premenných | Čas | Počet nodov | Počet po zredukovaní | Miera zredukovania |
| 10 | 172,14 | 297,17 | 42,12 | 85,01 |
| 11 | 262,25 | 500,96 | 49,59 | 89,24 |
| 12 | 439,39 | 927,85 | 61,87 | 92,44 |
| 13 | 739,13 | 1569,34 | 69,34 | 94,90 |
| 14 | 1249,33 | 2738,41 | 77,49 | 96,64 |
| 15 | 2212,23 | 5358,54 | 90,53 | 98,01 |
| 20 | 27306,18 | 83533,20 | 141,85 | 99,71 |

Graf nám znázorňuje priemerný počet nodov pre vytvorené stromy. Červena krivka je pre samotný BDD\_create. Vidíme, že BDD\_with\_best\_order nám naozaj vrátil výhodnejší strom.

Z grafu môžeme vidieť rýchlosť tvorenia stromov. Opäť je BDD\_with\_best\_order rýchlejší pre nájdený strom. Celkovo je však hľadanie najlepšieho stromu pomalšie, lebo sa generuje 100 stromov ktoré sa porovnávajú. Pri BDD\_create sa tvorí vždy iba jeden čiže spolu 200 a pri best order spolu 20 000 stromov.

BDD\_create\_with\_best\_order má vyššiu mieru zredukovania ako BDD\_create.

# **3. Výsledok**

Z grafu vidíme, že priemerná rýchlosť tvorenia BDD je O(2^n). Je to kvôli počtu premenných, lebo každou premennou sa pridáva ďalšia vrstva stromu, v ktorej sa nachádza dvojnásobok predošlej vrstvy. Priestorová zložitosť je pre najhorší prípad O(2^n) ale s redukciou to je O(log n). Z môjho grafu to nie je dobre vidieť, lebo som tam spravil skok medzi 15 a 20 premennými.