Adam Vrabeľ AIS ID: 116327

**FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ SLOVENSKÁ TECHNICKÁ UNIVERZITA**
Ilkovičova 2, 842 16 Bratislava 4

2022/ 2023

# Dátové štruktúry a algoritmy

## Zadanie č.2

Cvičiaci: **Ing. Martin Komák, PhD.**
**Čas cvičení: 16:00**

Vypracoval: **Adam Vrabeľ**
**AIS ID: 116327**

# Obsah

# Binárny rozhodovací diagram (BDD)

## ÚVOD

V tomto zadaní bolo mojou úlohou implementovať vlastné funkcie na vytvorenie a používanie binárneho rozhodovacieho diagramu (BDD). Implementoval som funkcie BDD_create(), BDD_create_with_best_order(), BDD_use a ostatné funkcie.
Binárny rozhodovací diagram je efektívny spôsob ako reprezentovať BOOLEOVSKÚ FUNKCIU.

## ŠTRUKTÚRA BDD

```
DSA - PROJEKT 2
Author: Adam Vrabeľ

21 usages  xvrabela1 *
public class BinaryDecisionDiagram {
    11 usages
    public int numberOfNodes;                    // POČET NODEOV V CELOM STROME (BEZ zeroNode A oneNode)
    9 usages
    public char[] poradieSpracovania = null;  // PORADIE AKO BUDEM POSTUPNE SPRÁCOVAVAŤ PREMENNÉ
    7 usages
    public Node root;               // KOREŇ STROMU


    // V CELOM STROME JE IBA JEDEN NULOVÝ A IBA JEDEN JEDNOTKOVÝ NODE
    3 usages
    Node zeroNode;          // NULOVÝ NODE
    3 usages
    Node oneNode;           // JEDNOTKOVÝ NODE


    // HASH-MAPA KDE SA UKLADAJU VYTVORENÉ NODE-y
    // PRI KAZDOM VYTVÁRANÍ SA SKONTROLUJE ČI NEEXISTUJE ROVNAKÝ NODE (S ROVNAKOU FORMULOU)
    //      AK EXISTUJE TAK SA NAPOJÍ TEN UŽ VYTVORENÝ
    //      AK NEEXISTUJE TAK AŽ VTEDY SA VYTVORÍ
    4 usages
    public HashMap<String, Node> hashMap = new HashMap<>();
```

Môj BDD (**BinaryDecisionDiagram**) obsahuje **počet nodeov** (**numberOfNodes**), **poradieSpracovania** reprezentované ako Array znakov, koreňový Node (**root**), nulový (**zeroNode**) a jednotkový (**oneNode**). BDD obsahuje aj **HashMapu<String, Node>** do ktorej ukladám novovytvorené jedinečné Node-y. V celom BDD sa nachádza vždy iba jeden nulový a jeden jednotkový Node (predvytvorený). Node reprezentuje jeden prvok BDD a je popísaný nižšie.

```java
public class Node {
    2 usages
    public String variable;    // podľa ktorého písmenka spracovávam
    5 usages
    String formula;     // samotný string ( napr. AB+AC+BD  )

    4 usages
    Node left;          // actual = 0
    4 usages
    Node right;         // actual = 1

    // KONŠTRUKTOR
    no usages  xvrabela1
    public Node(){
    }
    2 usages  xvrabela1
    public Node(String formula){
        this.formula = formula;
        left = null;
        right = null;
    }
    2 usages  xvrabela1
    public Node(String formula, String variable){
        this.variable = variable;
        this.formula = formula;
        left = null;
        right = null;
    }
}
```

Samotný Node (PRVOK BDD) obsahuje **variable** (písmeno podľa ktorého je Node spracovávaný), **formulu**, **ukazovateľ vľavo** (kde sa do formuly dosádza za spracovávané písmeno 0) a **ukazovateľ vpravo** (kde sa do formuly dosádza za spracovávané písmeno 1)

# POUŽÍVANÉ BOOLEOVSKÉ PRAVIDLÁ

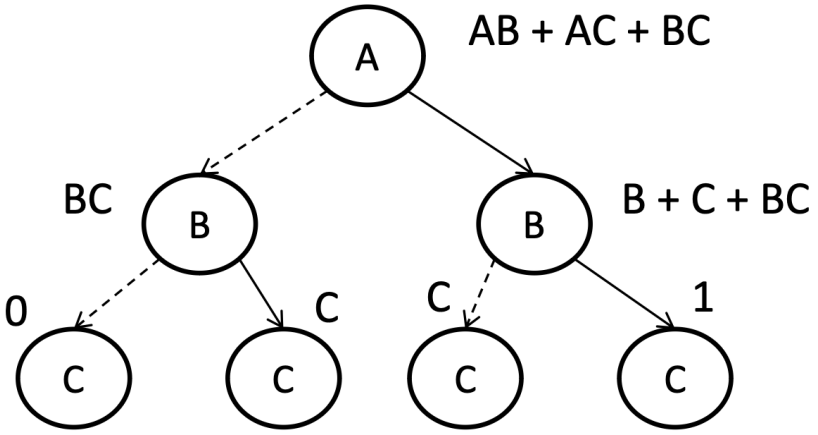
| | |
|---|---|
| **A + 0 = A**<br>**A . 1 = A** | **A + !A = 1**<br>**A . !A = 0** |
| **A . 0 = 0**<br>**A + 1 = 1** | **A + A = A**<br>**A . A = A** |

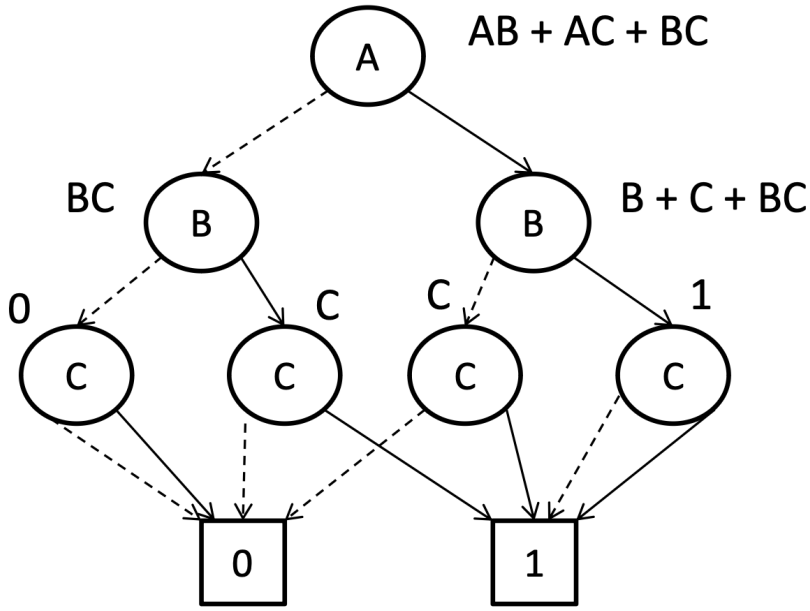
# REDUKCIE

V mojom programe používam 2 typy redukcií. Keďže môj BDD udržiavam redukovaný počas vytvárania, moje riešenie je priestorovo a časovo efektívnejšie.

Vysvetlenie na príklade:

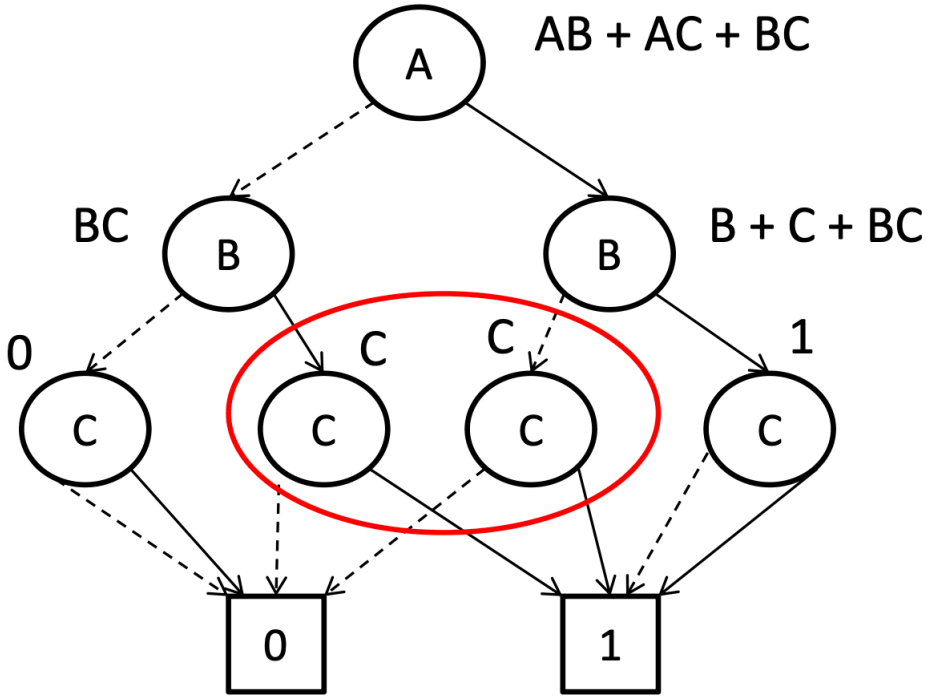
BDD bez redukcií pre formulu „AB+AC+BC“ s poradím „ABC“ by vyzeral následovne.

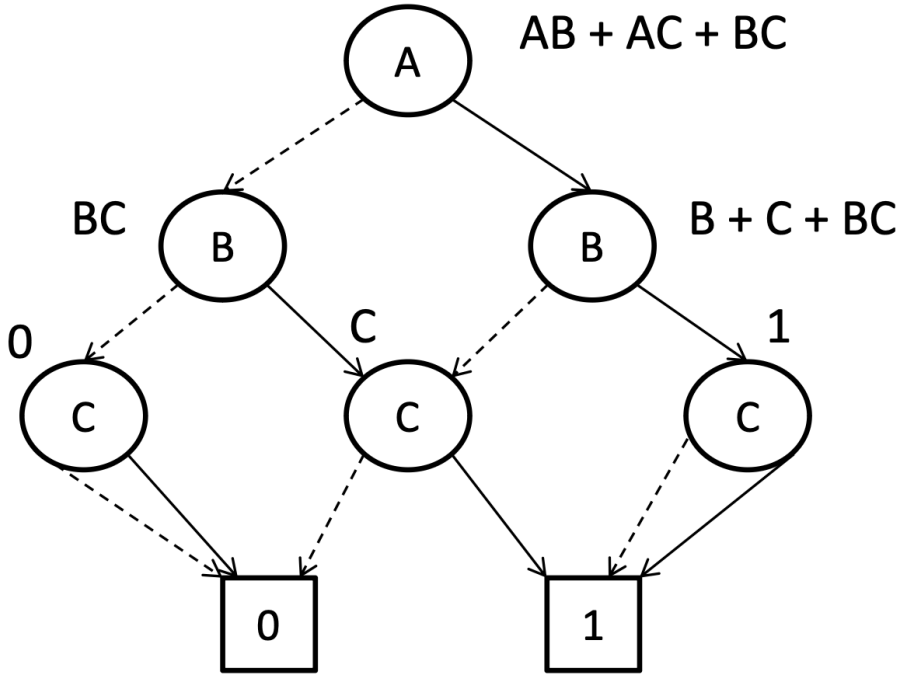


# REDUKCIA TYPU I

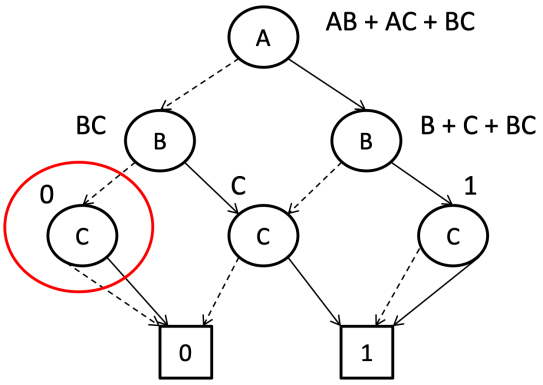
V BDD si držím informácie pomocou HashMapy o vytvorených Nodeoch. Kde HashMapa je <String formula, Node node>, ak sa pridávaná formula zhoduje s formulou v hashmape, tak sa na dané miesto napojí už vytvorená.

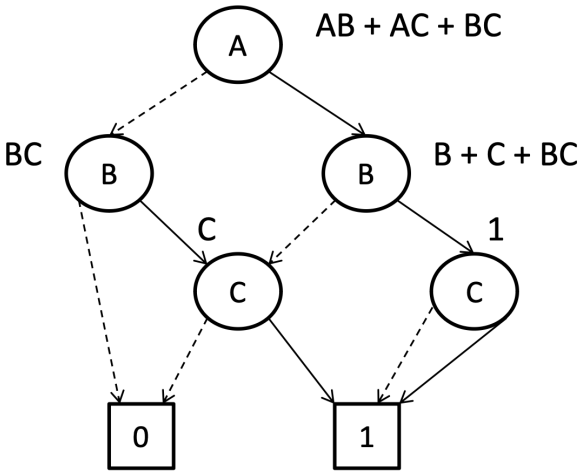


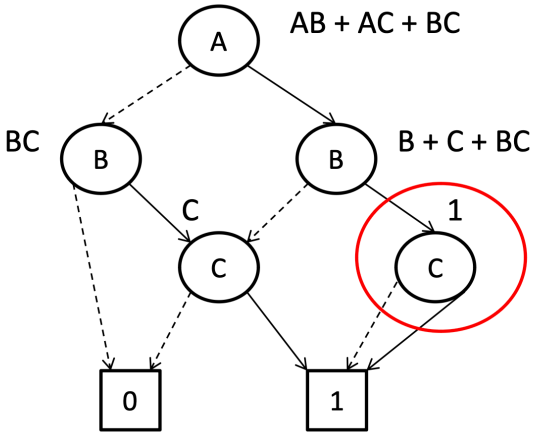



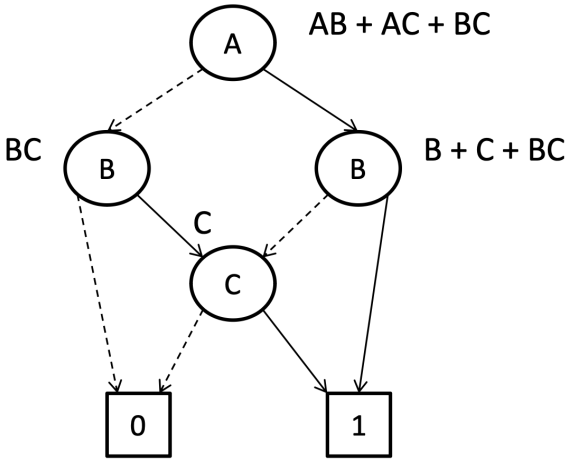
# REDUKCIA TYPU S

V celom strome existuje vždy iba jeden oneNode a zeroNode. Ak sa pri vytváraní stromu formula zkráti na „1“ resp. „0“ tak sa vždy napojí na predvytvorený oneNode resp. zeroNode



# BDD_CREATE

Funkcia BDD_create(String formula, String poradie) slúži na vytvorenie redukovaného binárneho rozhodovacieho diagramu, ktorý opisuje zadanú Booleovskú funkciu **„formula“**. Poradie spracovávania formule v BDD je definované reťazcom **„poradie“**.
Redukciu vykonávam počas vkladania.

- Funkcia dostane počiatočnú formulu (napr ABAA+BC+!DA) s poradím „ABCD“
- Počiatočnú formulu si pomocou funkcie zjednodusFormulu() zjednoduším. Táto funkcia vymaže duplicitné znaky medz +, zoradí premenné vrámci podstringov rozdelených cez + abecedne, a negácie umiestni pred písmeno kam abecedne patrí. Výsledná formula z funkcie zjednodusFromulu() je „AB+BC+A!D“
- Následuje kontrola či zjednodušená formula je „1“ alebo „0“, ak nieje tak prechádzam na funkciu BDD_insertNode(formula, indexZanorenia), ktorá rekurzívne vkladá nové prvky do BDD. (indexZanorenia reprezentuje index pola v poradí (ktorú premennú spracovávam))

**BDD_insertNode(formula, indexZanorenia)** v tejto funkcii následne prechádzam poradím a zjednodušujem formulu (dosádzam do aktuálnej formule za premennú určenú podľa indexuZanorenia 0 alebo 1 podľa toho či idem vľavo alebo vpravo v danom Node)

Dosádzanie 0 alebo 1 realizuje funkcia dosadDoFormuly() táto funkcia dosadí za daný znak dané číslo a formulu pomocou Booleovských operácií zjednoduší.

**BDD_insertNode** je rekurzívna funkcia, ktorá vola samú seba s indexZanorenia++ a dosadenými 0 za spracovávanú premennú (vľavo) respektíve dosadenými 1 za spracovanú premennú (vpravo). V prípade, že formula ešte nieje v HashMape vytvorených prvkov, nieje to „1“ alebo „0“ tak vráti novo vytvorený Node a pridá ho do HashMapy. Rekurzívne napojí všetky jedinečné Node do stromu, kde listami stromu je oneNode a zeroNode.

Funkcia BDD_create nastaví pre inštanciu **BinaryDecisionDiagram** všetky jeho parametre vrátane ukazovateľa na začiatok (root) BDD.

# BDD_CREATE_WITH_BEST_ORDER

Tátu funkcia nájde čo najlepšie poradie (vrámci skúšaných možností) pre zvolenú Booleovskú formulu v BDD.
BDD_create_with_best_order(String formula):

- Ako prvé si z formuly pomocou regex výrazov vytiahnem jedinečné písmená a zoradím abecedne, kde dostanem prvotné poradie ktoré je zložené zo všetkých písmien, ktoré sa nachádzajú vo formuli a je abecedne usporiadané
- Následne vytváram BDD z prvotného poradia pomocou funkcie BDD_create()
- V cykle posúvam poradie a volám BDD_create() toľko krát, aké je dlhé poradie
- Na konci funkcie vrátim ten BDD ktorý má najmenší počet Nodeov (je najoptimálnejší)

AK MÁM NAPR FORMULU AB+CD
PRVOTNÉ PORADIE : ABCD A NÁSLEDNE HO POSÚVAM: BCDA, CDAB, DABC

# BDD_USE

Táto funkcia slúži na použitie vytvoreného BDD. Argumentom tejto funkcie je String s poradím čísel ako sa majú dosádzať do stromu. Poradie týchto čísel reprezentujé hodnotu pre premenné v poradí BDD.

Pre BDD s formulou (AB+C) a s poradím [A,C,B]
BDD_USE(010)  A=0, C= 1, B=0

BDD_USE prechádza vytvoreným BDD ak je zadané číslo na indexe rovné 0, ide v strome vľavo ak je rovné 1, ide v strome vpravo. Ak sa písmeno na dosadenie v aktuálnej formuli nenachádza, tak prechádza na dalšie. Ak narazí na oneNode alebo zeroNode, vráti 1 respektíve 0 inak vráti E ako chybu. (nedošlo k výsledku, čísla na dosadenie niesu dostatočne dlhé)

# ČASOVÁ NÁROČNOSŤ

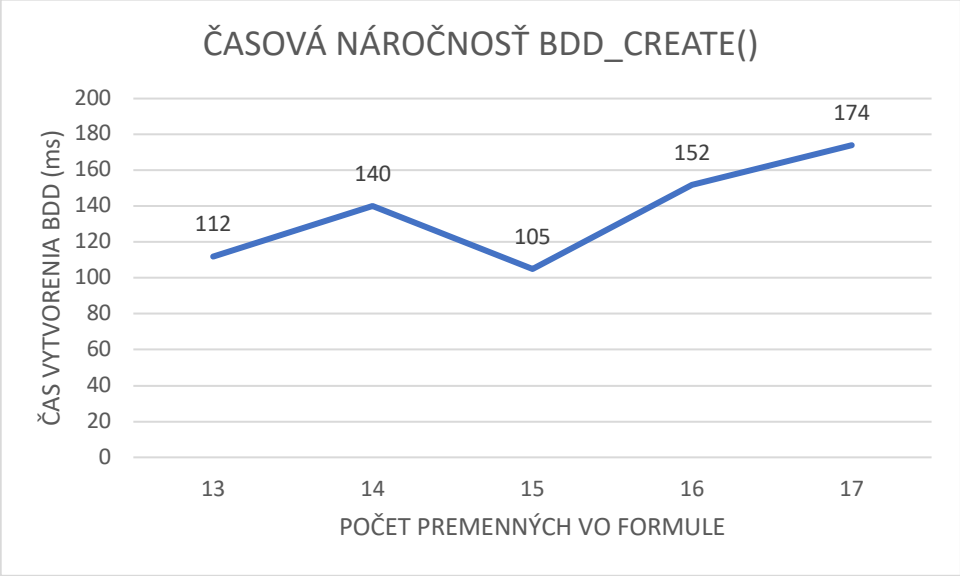
Testoval som v prostredí intelliJ IDEA na počítači s Apple M1 PRO chipom.

# ČASOVÁ NÁROČNOSŤ BDD_CREATE

Testovanie časovej náročnosti BDD_create funkcie.
Dopredu som si vygeneroval formuly pre 13, 14, 15, 16 a 17 premenných pomocou funkcie vygenerujFormulu() aby generovanie neovplyvnilo meranie času.

Následne pre každú vygenerovanú formulu som s rovnakým (abecedným) poradím zavolal 1000-krát BDD_create() a meral čas.

Odhadovaná časová náročnosť funkcie BDD_create je O(n^2) ale všetko závisí od vygenerovanej formule a určeného poradia, čo má značný vplyv na rýchlosť vytvárania.



# ČASOVÁ NÁROČNOSŤ BDD_CREATE_WITH_BEST_ORDER

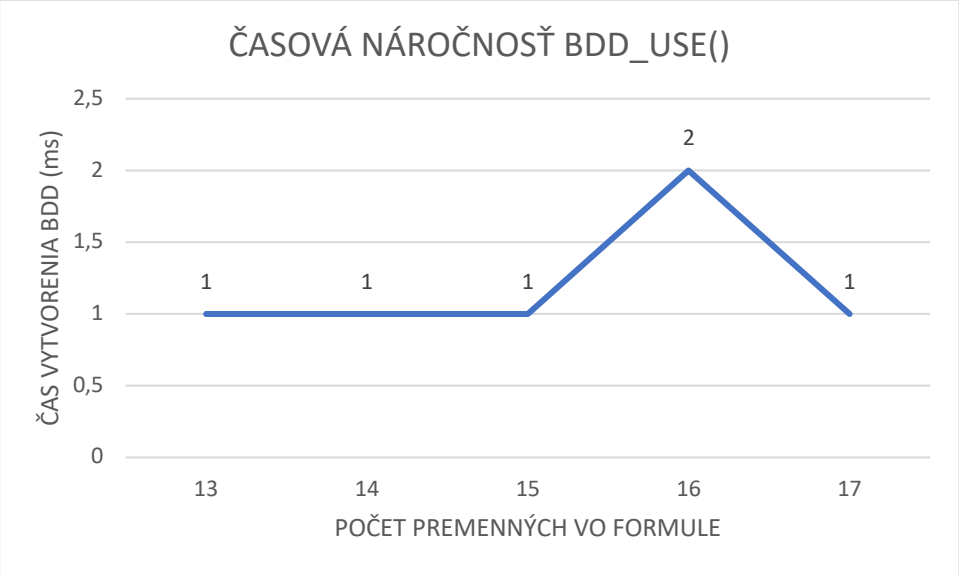
Časová náročnost BDD_CREATE_WITH_BEST_ORDER je priamo závisla na počte premenných. Ak máme formulu s 3 premennými tak časová náročnosť BDD_CREATE_WITH_BEST_ORDER je 3*časová náročnosť BDD_CREATE() pretože pri best order volám funkciu toľko krát koľko je premenných vo formuli a posúvam poradie.

Nech n je počet premenných vo formuli, tak odhadovaná časová náročnosť funkcie BDD_CREATE_WITH_BEST_ORDER je n*( O(n^2) )

# ČASOVÁ NÁROČNOSŤ BDD_USE

Testovanie časovej náročnosti BDD_use funkcie.
Dopredu som si vygeneroval formuly pre 13, 14, 15, 16 a 17 premenných pomocou funkcie vygenerujFormulu() aby generovanie neovplyvnilo meranie času. Vytvoril im pomocou BDD_create() BDD.

Následne pre každú vygenerovanú formulu som zavolal 1000-krát BDD_use() s dopredu určeným poradím a meral čas.



# PRIESTOROVÁ NÁROČNOSŤ

Na meranie veľkosti využitej pamäte využívam funkcie na získanie celkovej pamäte, ktorú si program vyhradí na heap. Získam taktiež free pamäť programu. Ich rozdielom je výsledok využitej pamäte programu.

Testovanie priestorovej zložitosti som aplikoval na test2() ktorý vytvára 100 BDD pomocou BDD_CREATE_WITH_BEST_ORDER pre formulu s 13 premennými

Celková pamäť: 260MB
Použitá pamäť:  22MB

```
// PRIESTOROVÁ NÁROČNOSŤ ////////////////////////////////////////////////////////////
long total = Runtime.getRuntime().totalMemory();
long used = Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory();

System.out.println("Celková pamat: " + total/1024/1024 + "MB");
System.out.println("Pouzitá pamat: " + used/1024/1024 + "MB");
////////////////////////////////////////////////////////////////////////////////////
```

# TESTOVANIE

Testy sa nachádzaju v triede TESTS.java Metóda main v tejto triede je spustiteľná.

## test()

V tomto teste som overil správnosť môjho vytvoreného BDD pomocou BDD_USE pre formulu (AB+C) s poradím [ABC]
Volaním BDD_USE pre všetky možnosti poradia čísel na dosadenie.

## test1()

Tento test slúži na testovanie základnej funkcionality BDD.
Určím si testovaciu formulu, poradie a čísla na dosadenie do BDD.

Vytvorím 2 BDD ( jeden so zadaným poradím (neoptimálny) a jeden pomocou BCC_CREATE_WITH_BEST_ORDER (optimálny) )
Pre vytvorené BDD vypíšem informácie: počet Nodeov pre každý BDD, reduction rate pre každý BDD.

A ako posledné porovnám výsledky BDD_USE a priameho dosadenia do formuly bez použitia BDD (dosadenie „nahrubo“ vykoná funkcia nahruboDosadDoFormuly).

Funkcia nahruboDosadDoFormuly má parametre: formula, poradie, cislaNaDosadenie). Dosadí do formuly postupne čísla v poradí a vráti výsledok podobne ako BDD_USE.

| | **NEOPTIMALNY BDD** | **OPTIMALNY BDD** |
|---|---|---|
| **FORMULA** | AB+CD+EF+GH | AB+CD+EF+GH |
| **PORADIE** | ACGBEFDH | ABCDEFGH |
| **POČET NODEov** | 24 | 10 |
| **REDUCTION RATE (%)** | 90,66 | 96,11 |

## test2()

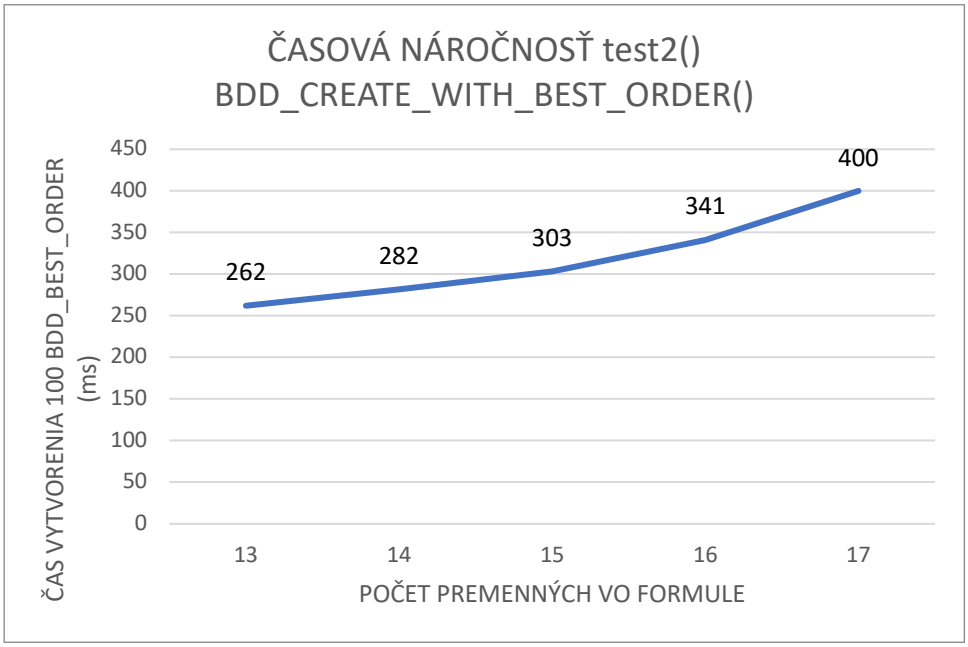
Tento test vytvorí 100 BDD STROMOV, jednotlivé stromy sú vytvorené pomocou funkcie BDD_CREATE_WITH_BEST_ORDER s dopredu určeným počtom premenných.

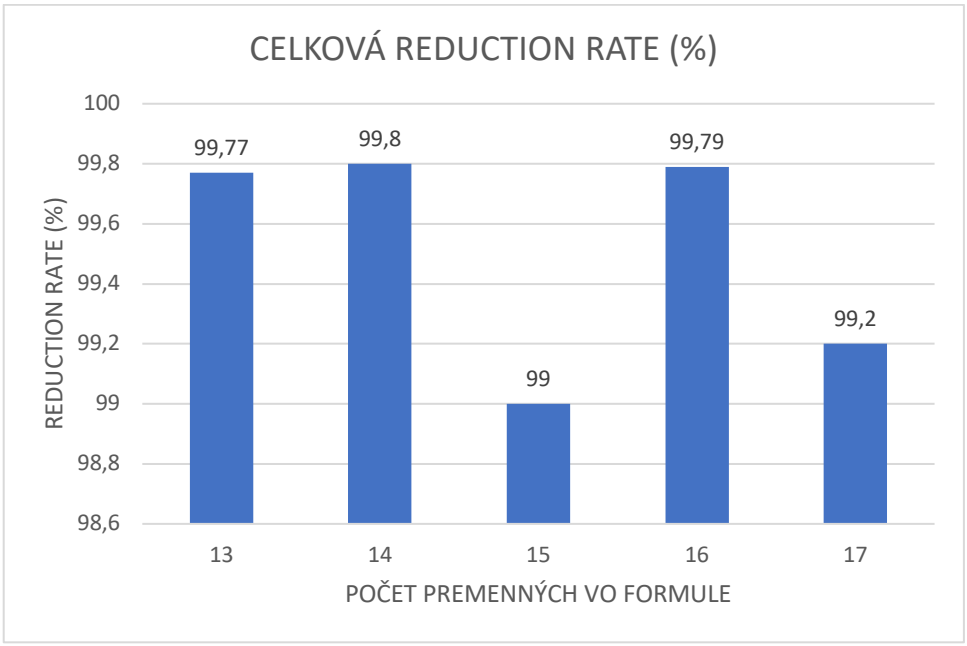
V každej iterácii zo 100 sa vygeneruje náhodna formula s dopredu definovaným počtom premenných, pozostávajúca z daných písmen, + a !

V každom vytvorení stromu cez BDD_CREATE_WITH_BEST_ORDER sa zavolá BDD_CREATE toľko krát aký je počet premenných.

Zároveň sa počíta reduction rate pre každý vytvorený strom (BEST_ORDER) a celkový reduction rate pre všetkých 100 stromov (BEST_ORDER).

**Funkcia vygenerujFormulu(int pocetPremennych)** vráti String formula ktorá je zložená z 5 až 10 súčtov, vrámci súčtov sa generuje náhodne početNásobení v rozmedzí (pocetPremennych/2 až 2*pocetPremennych) čo predstavuje počet vygenerovaných písmien v jednotlivých súčtoch, jednotlive písmená sa generujú od A do (A+pocetPremenných) a šanca na negáciu pre každé písmeno je ¼.



```
CISLO TESTU [63]
VYGENEROVANA FORMULA: [!F!B!EJ!GECH!EC!IMGGEBM!HE!EK+DAMMEBD!KA+HMH!IFF+GA!I!BF!CL+M!KHGLF!H!M!DAKM!I!HCB!F!BI]
MAXIMALNY POCET NODEOV: [8193]
POCET NODEOV: [25]        PORADIE V STROME: [A, B, C, D, E, F, G, H, I, J, K, L, M]
AKTUALNY BDD  [REDUCTION RATE: 99,69 %]
DO BDD [!F!B!EJ!GECH!EC!IMGGEBM!HE!EK+DAMMEBD!KA+HMH!IFF+GA!I!BF!CL+M!KHGLF!H!M!DAKM!I!HCB!F!BI] S PORADIM [ABCDEFGHIJKLM] DOSADZAM [0110001111000]
[VYSLEDOK DOSADENIM: 0]  [VYSLEDOK BDD_USE: 0]

CISLO TESTU [64]
VYGENEROVANA FORMULA: [JI!JJFA!FMI!MIAGL!DI!D!L!IC!IKF!KG+DGGBICGB!A!EIIMJFC+ICHEDC!DF!KLL!HJJK!GKB!KB+FKDJBDBIJC!BIHJM+DKJ!MMLEABLBCC!DAIF!BL!KJ!GC+A!AHBLB!HFLJC!HLJ+A!AIMMA!DD!B!L!LF!C
MAXIMALNY POCET NODEOV: [8193]
POCET NODEOV: [23]        PORADIE V STROME: [A, B, C, D, E, F, G, H, I, J, K, L, M]
AKTUALNY BDD  [REDUCTION RATE: 99,72 %]
DO BDD [JI!JJFA!FMI!MIAGL!DI!D!L!IC!IKF!KG+DGGBICGB!A!EIIMJFC+ICHEDC!DF!KLL!HJJK!GKB!KB+FKDJBDBIJC!BIHJM+DKJ!MMLEABLBCC!DAIF!BL!KJ!GC+A!AHBLB!HFLJC!HLJ+A!AIMMA!DD!B!L!LF!C+B!D!HJAGCK] S
[VYSLEDOK DOSADENIM: 0]  [VYSLEDOK BDD_USE: 0]

CISLO TESTU [65]
VYGENEROVANA FORMULA: [CJJAJ!ALL!DFEKLBM!DEBCG!ICJM+KJFCCJFGKL!KD+G!MBD!EAJG!ELJ+FD!LKDCCCLJFCDHL!G!F!MJJ!G+CF!EJAAB!E!J!MHIL+LLFHKBLIHL!F!LA!L!GI+CIJ!DKFCE!EGFIIM!LELLAI!EADL+C!MKCIM!GM
MAXIMALNY POCET NODEOV: [8193]
POCET NODEOV: [22]        PORADIE V STROME: [I, J, K, L, M, A, B, C, D, E, F, G, H]
AKTUALNY BDD  [REDUCTION RATE: 99,73 %]
DO BDD [CJJAJ!ALL!DFEKLBM!DEBCG!ICJM+KJFCCJFGKL!KD+G!MBD!EAJG!ELJ+FD!LKDCCCLJFCDHL!G!F!MJJ!G+CF!EJAAB!E!J!MHIL+LLFHKBLIHL!F!LA!L!GI+CIJ!DKFCE!EGFIIM!LELLAI!EADL+C!MKCIM!GMIGBDM!GKCIIADJ+
[VYSLEDOK DOSADENIM: 0]  [VYSLEDOK BDD_USE: 0]
```