Datové štruktúry a algoritmy

**Zadanie 3 – Binárne rozhodovacie diagramy**

Emma Macháčová

**Meno cvičiaceho** : Ing. Dominika Dolhá

**Čas cvičení** : pondelok 9:00

**Dátum vytvorenia** : 13. apríl 2021

**Obsah**

[Cieľ práce 1](#_Toc69313374)

[Opis riešenia 2](#_Toc69313375)

[Funkcia BDD\_create 3](#_Toc69313376)

[Funkcia BDD\_reduce 4](#_Toc69313377)

[Funkcia BDD\_use 5](#_Toc69313378)

[Testy 6](#_Toc69313379)

[Testovacia funkcia 6](#_Toc69313380)

[Výsledky testov 7](#_Toc69313381)

Cieľ práce

Cieľom projektu v rámci tohto zadania je vytvoriť program, ktorý bude vedieť vytvoriť, redukovať a použiť dátovú štruktúru BDD (Binárny Rozhodovací Diagram) so zameraním na využitie pre reprezentáciu Booleovských funkcií.

Konkrétne je cieľom implementovať **tieto funkcie**:

* BDD \***BDD\_create**(BF \*bfunkcia);
* int **BDD\_reduce**(BDD \*bdd);
* char **BDD\_use**(BDD \*bdd, char \*vstupy);

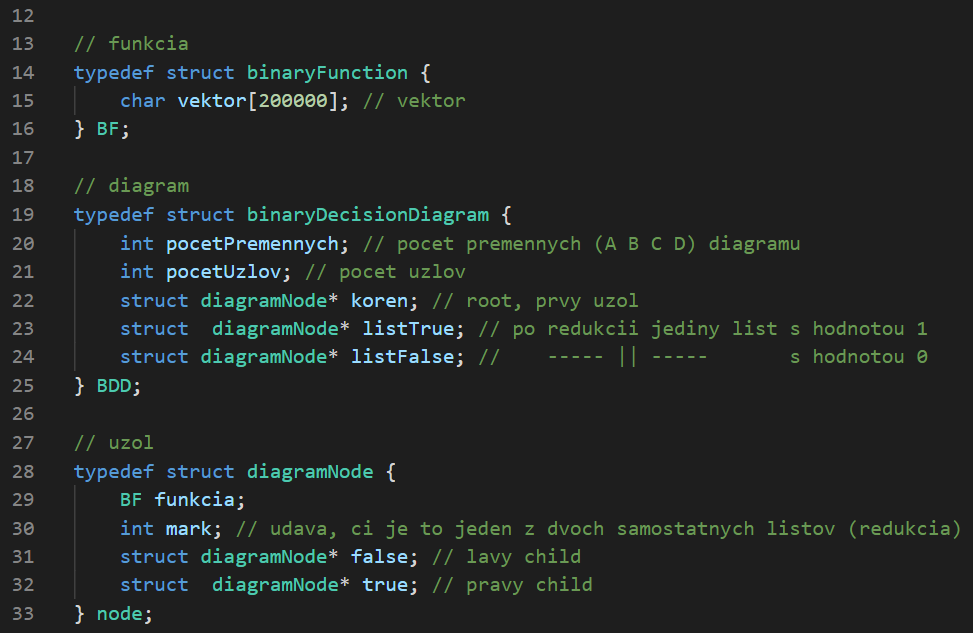
**Funkcia BDD\_create** má slúžiť na zostavenie úplného (t.j. nie redukovaného) binárneho rozhodovacieho diagramu, ktorý má reprezentovať/opisovať zadanú Booleovskú funkciu (vlastná štruktúra s názvom BF), na ktorú ukazuje ukazovateľ bfunkcia, ktorý je zadaný ako argument funkcie BDD\_create. Návratovou hodnotou funkcie BDD\_create je ukazovateľ na zostavený binárny rozhodovací diagram, ktorý je reprezentovaný vlastnou štruktúrou BDD.

**Funkcia BDD\_reduce** má slúžiť na redukciu existujúceho (zostaveného) binárneho rozhodovacieho diagramu. Aplikovaním tejto funkcie sa nesmie zmeniť Booleovská funkcia, ktorú BDD opisuje. Cieľom redukcie je iba zmenšiť BDD odstránením nepotrebných (redundantných) uzlov. Funkcia BDD\_reduce dostane ako argument ukazovateľ na existujúci BDD (bdd), ktorý sa má redukovať. Redukcia BDD sa vykonáva priamo nad BDD, na ktorý ukazuje ukazovateľ bdd. Návratovou hodnotou funkcie BDD\_reduce je číslo typu int (integer), ktoré vyjadruje počet odstránených uzlov. Ak je toto číslo záporné, vyjadruje nejakú chybu (napríklad ak BDD má ukazovateľ na koreň BDD rovný NULL).

**Funkcia BDD\_use** má slúžiť na použitie BDD pre zadanú (konkrétnu) kombináciu vstupných premenných Booleovskej funkcie a zistenie výsledku Booleovskej funkcie pre túto kombináciu vstupných premenných. V rámci tejto funkcie „prejdete“ BDD stromom smerom od koreňa po list takou cestou, ktorú určuje práve zadaná kombinácia vstupných premenných. Argumentami funkcie BDD\_use sú ukazovateľ s názvom bdd ukazujúci na BDD (ktorý sa má použiť) a ukazovateľ s názvom vstupy ukazujúci na začiatok poľa charov (bajtov). Práve toto pole charov/bajtov reprezentuje konkrétnu kombináciu vstupných premenných Booleovskej funkcie.

Opis riešenia

V projekte pracujem s troma **vlastnými definovanými štruktúrami**: binaryFunction, binaryDecisionDiagram a diagramNode.

* Štruktúra **binaryFunction** reprezentuje binárnu funkciu. Obsahuje pole, v ktorom sa uchováva vektor (sekvenciu 0/1), ktorým je daná.
* Štruktúra **diagramNode** predstavuje jeden uzol binárneho rozhodovacieho diagramu. Jeho zložkami sú binárna funkcia, a ukazovatele na nasledujúce uzly (ľavý a pravý).
* Štruktúra **binaryDecisionDiagram** reprezentuje už zostavený binárny rozhodovací diagram. Obsahuje počet premenných, počet uzlov a ukazovateľ na koreň (prvý uzol).

Okrem štruktúr a funkcií zo zadania využívam aj vlastné **pomocné funkcie**:



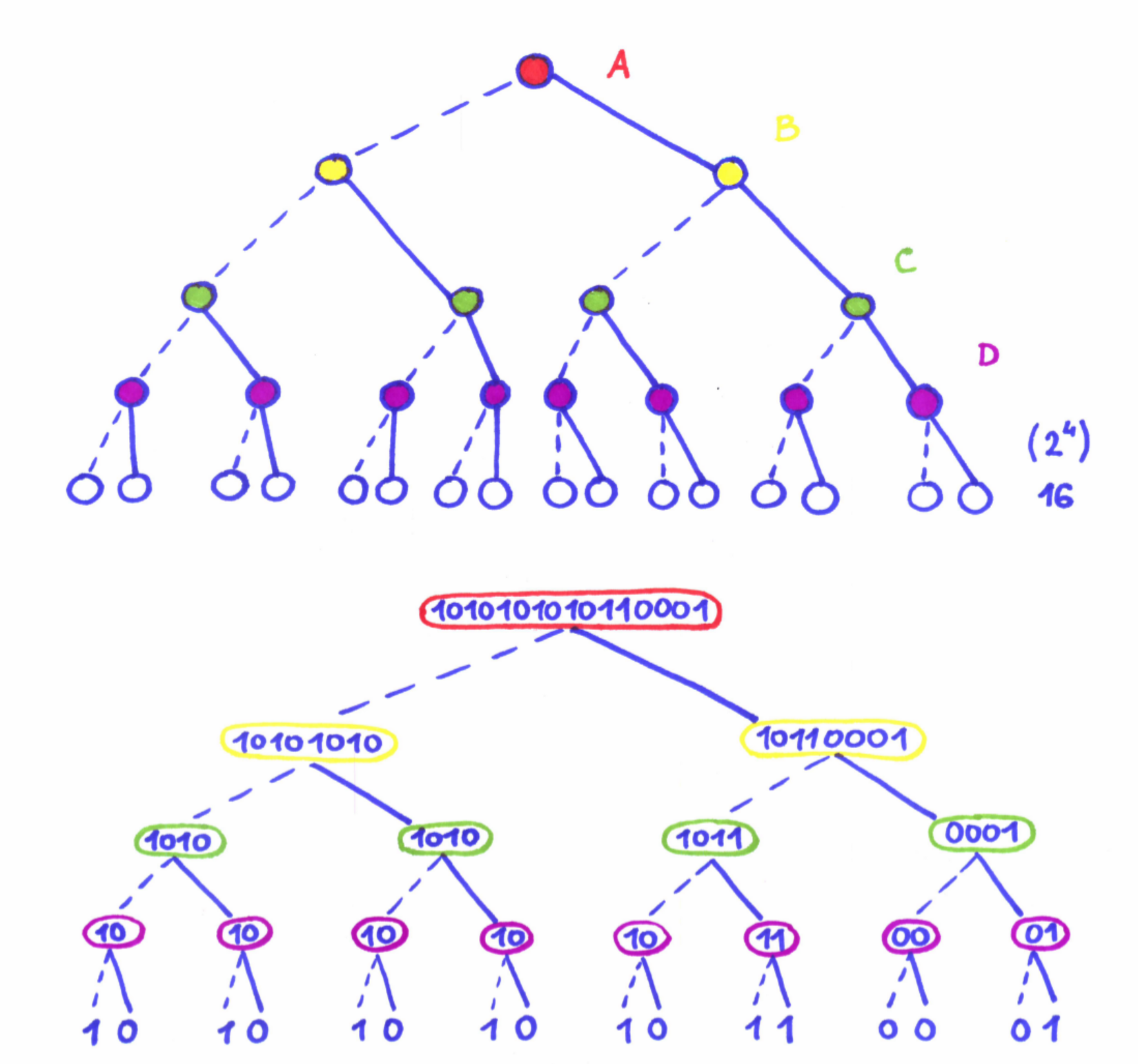
**Diagram je vytváraný** pomocou vstupnej funkcie (vektora) z hora nadol, postupným delením (vetvením) funkcie.

Funkcia BDD\_create

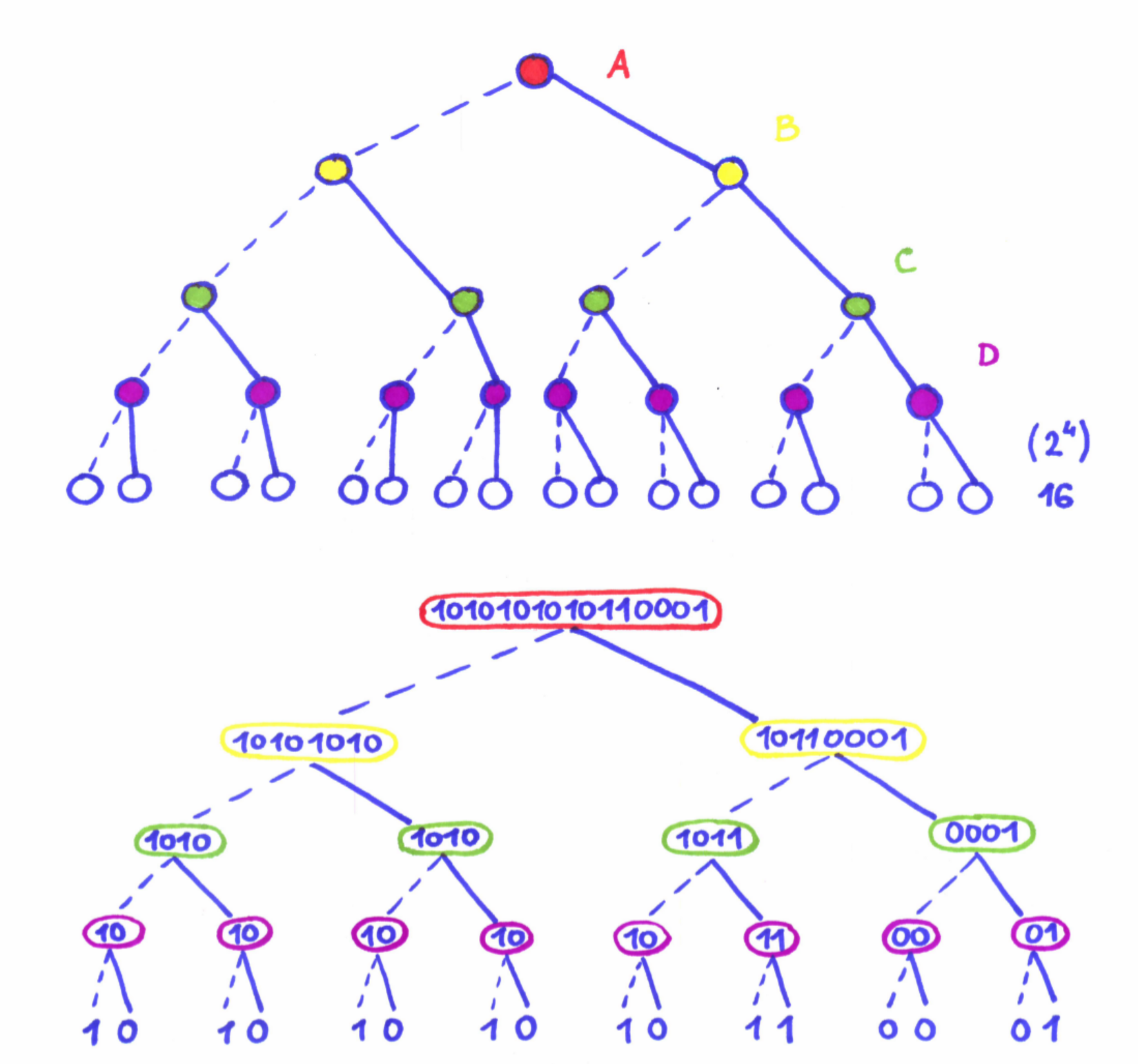
Funkcia má slúžiť na zostavenie úplného binárneho rozhodovacieho diagramu, ktorý má reprezentovať/opisovať zadanú Booleovskú funkciu. Návratovou hodnotou funkcie BDD\_create je ukazovateľ na zostavený binárny rozhodovací diagram.

**Funkcia najskôr** alokuje koreň (prvý uzol), štruktúru diagramu, a vstupný parameter binárnej funkcie vloží do (teraz alokovaného) prvého uzla.

Program ďalej pomocnou funkciou **getPocetPremennych** vypočíta to, koľko premenných má daná funkcia. Tento počet je logaritmom so základom dva z počtu outputov (listov) funkcie (ak má vstupný vektor dĺžku 16, počet listov diagramu je rovnako 16, a logaritmus so základom 2 vráti 4 – čo je zodpovedajúce počtu premenných funkcie).



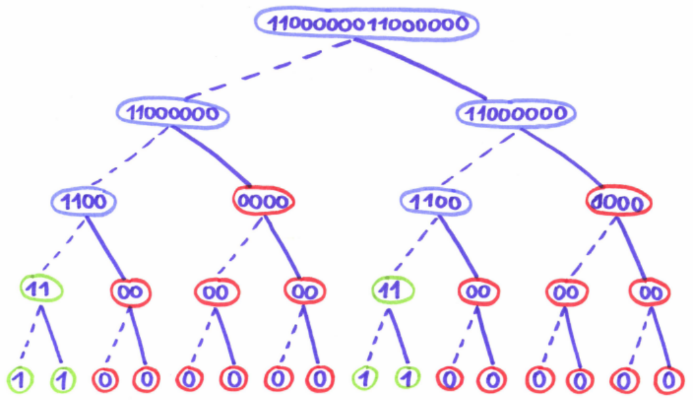
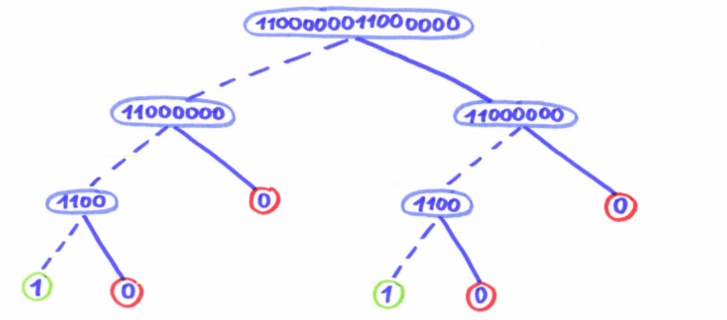
Prvý uzol sa nastaví ako koreň diagramu, a počet uzlov diagramu sa nastaví na 1. Následne sa na rozvetvenie diagramu volá rekurzívna funkcia **rozdel.** Táto funkcia od koreňu k listom postupne každému uzlu vytvorí dvoch potomkov, a pomocou funkcií **leftHalf** a **rightHalf** rozdelí pôvodný vektor na pravú a ľavú polovicu, ktoré im pridelí ako ich funkciu. Rekurzia pokračuje dovtedy, kým je ešte možné vektor deliť na dve polovice.

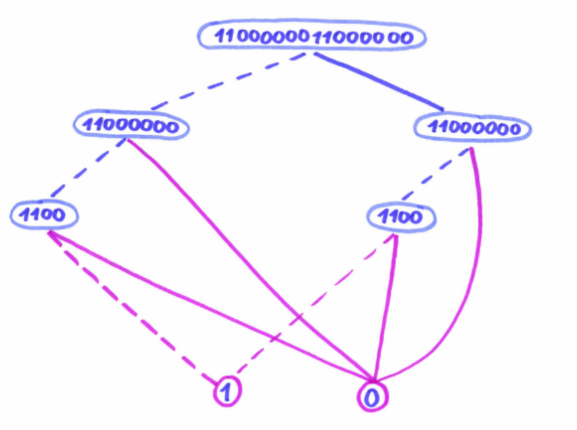


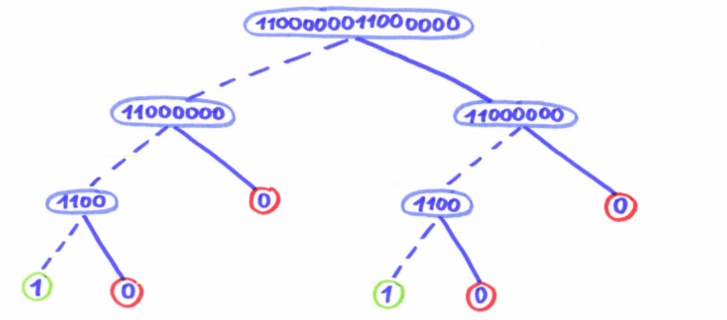
Keď sa ukončí delenie, funkcia vráti ukazovateľ na vytvorený binárny rozhodovací diagram.

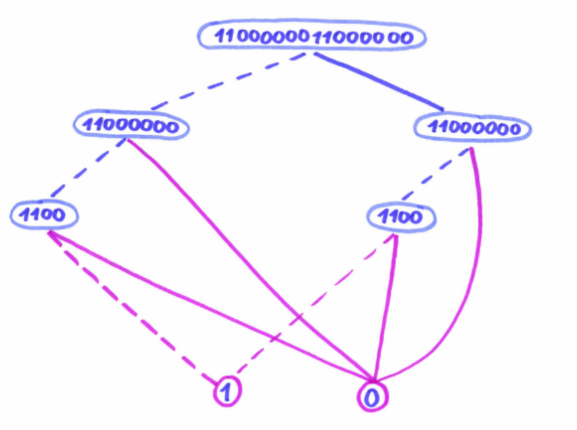
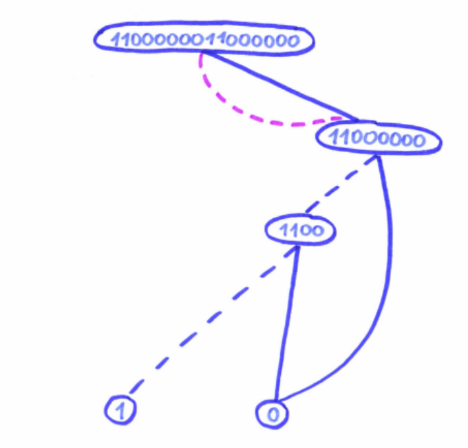
Funkcia BDD\_reduce

Funkcia má slúžiť na redukciu existujúceho binárneho rozhodovacieho diagramu. Cieľom redukcie je zmenšiť BDD odstránením nepotrebných uzlov. Funkcia BDD\_reduce dostane ako argument ukazovateľ na existujúci BDD (bdd), ktorý sa má redukovať. Redukcia BDD sa vykonáva priamo nad BDD, na ktorý ukazuje ukazovateľ bdd. Návratovou hodnotou funkcie BDD\_reduce je číslo typu int (integer), ktoré vyjadruje počet odstránených uzlov.

**Funkcia najskôr** overí to, či bol diagram vytvorený. Následne získa koreň diagramu, a zavolá s ním pomocnú rekurzívnu funkciu **najdiPrebytocne**. Tá vyhodnotí prebytočné uzly (také, čo obsahujú len 1 alebo len 0), a odstráni ich.

Po odstránení zbytočných uzlov sa volá pomocná rekurzívna funkcia **spojKonce**, ktorá nahradí všetky listy stromu dvoma špeciálnymi, na ktoré ukazuje aj štruktúra BDD.



Ako posledná sa volá pomocná funkcia **najdiSymetricke,** ktorá funguje podobne ako funkcia najdiPrebytocne, ale hľadá uzly, ktoré sú symetrické (11001100, 0000110000001100..), a odstráni jedného z ich potomkov.

Funkcia BDD\_use

Funkcia má slúžiť na použitie BDD pre zadanú (konkrétnu) kombináciu vstupných premenných Booleovskej funkcie a zistenie výsledku Booleovskej funkcie pre túto kombináciu vstupných premenných.

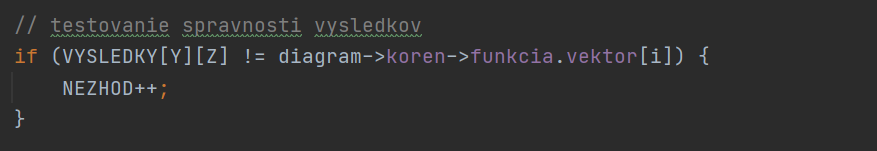
**Funkcia najskôr** kontroluje, či existuje binárny diagram. Pokiaľ nie, vráti „x“.

Ak diagram existuje, zavolá pomocnú rekurzívnu funkciu **trasa**. Tá rekurzívne prejde strom od koreňa po listy. To, do ktorej strany sa bude pohybovať, závisí od premennej lr (left / right), ktorá nadobúda hodnotu pre danú premennú na tej úrovni stromu, podľa poľa vstupu – „string“ 0/1 (čím hlbšie v strome tým vyšší index – čo úroveň to index). Pole vstupu generuje buď funkcia **vytvorVstup** (jeden náhodný vstup pre daný počet premenných diagramu), alebo funkcia **otestujVsetkyMoznosti** (rekurzívna funkcia, vytvorí postupne všetky možné vstupy pre daný počet prvkov diagramu).

Ak bol strom redukovaný a funkcia narazí na list pred prečítaním celého vstupu, vráti tento list (preto že ďalšie čítanie vstupu by bolo zbytočné).

**Po ukončení funkcie trasa**, funkcia BDD\_use overí hodnotu listu, ku ktorému funkcia trasa prišla. Pokiaľ je táto hodnota 0, funkcia vráti 0. Ak je to 1, funkcia vráti 1.

**Správnosť výsledkov sa overuje porovnávaním ich s hodnotou vektora na príslušnom indexe.**



Testy

V rámci testovanie bolo potrebné náhodným spôsobom generovať Booleovské funkcie, podľa ktorých sa vytvárajú BDD pomocou funkcie BDD\_create. Vytvorené BDD je potrebné následne zredukovať funkciou BDD\_reduce a nakoniec overiť 100% funkčnosť zredukovaných BDD opakovaným (iteratívnym) volaním funkcie BDD\_use s postupným použitím všetkých možných kombinácií vstupných premenných.

Počet premenných v rámci testovania BDD by mal byť minimálne 13. Počet Booleovských funkcií / BDD diagramov by mal byť minimálne 2000. V rámci testovania sa tiež vyhodnocuje percentuálna miera zredukovania BDD (t.j. počet odstránených uzlov / pôvodný počet uzlov).

Výsledky testovania majú obsahovať (priemernú) percentuálnu mieru zredukovania BDD a (priemerný) čas vykonania funkcií. Zhodnotenie má obsahovať odhad výpočtovej (časovej) a priestorovej (pamäťovej) zložitosti algoritmu.

Testovacia funkcia

Program obsahuje dve testovacie funkcie: **velkyTest, test** a **malyTest**.

**Malý test** funguje ako ukážka na strome, ktorý používam ako ilustračný aj v tejto dokumentácií. Je menší a preto na ňom ide jednoducho skontrolovať správnosť programu.

**Veľký test** funguje podľa zadania – pre 14 premenných, vykoná 2000 testov (čo test to iná funkcia – náhodne generovaná funkciou vytvorFunkciu), a v rámci týchto 2000 testov otestuje všetky možnosti vstupov pre premenné diagramu (214 vstupov).

Tento test najskôr vytvorí funkciu, a podľa nej sa vytvorí diagram. Vypíšu sa kontrolné správy, ktoré je možné aj uložiť do súboru.

Následne sa pre neredukovaný diagram otestujú všetky možné vstupy (2n), ktoré boli vytvorené dopredu funkciou **generujVsetkyMoznosti**. Výsledky testov sa uložia do poľa.

Po otestovaní neredukovaného diagramu prebehne redukcia funkciou BDD\_reduce, a v redukovanom diagrame sa rovnako otestujú všetky vstupy, a výsledky sa tiež zapíšu do poľa.

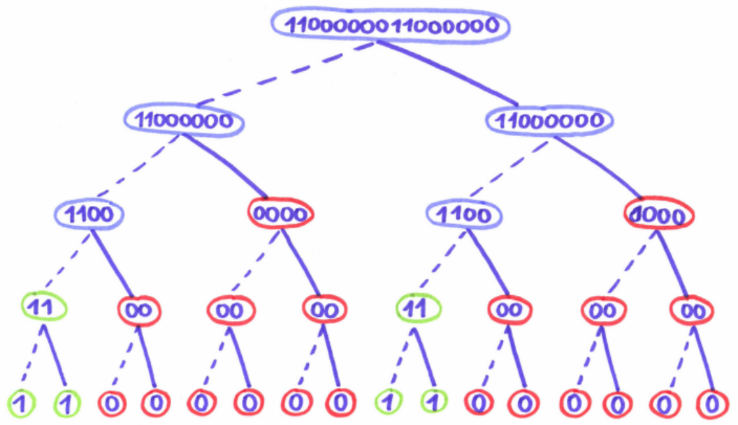
Po tom, čo prebehnú testy v oboch diagramoch, porovnajú sa výsledky a počet nezhôd sa zaráta. Diagram sa po dokončení každého cyklu uvoľní.

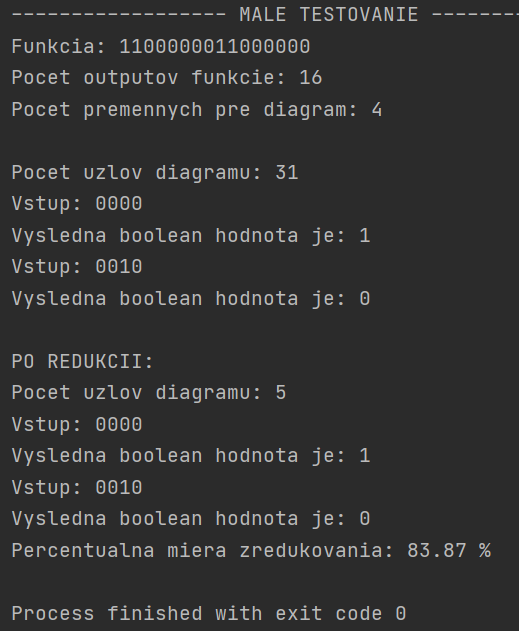
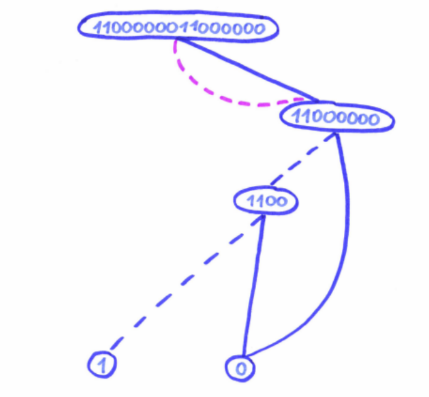
Toto prebehne toľkokrát, koľko testov užívateľ zadá (alebo 2000 preddefinovaných.. počet testov aj počet premenných diagramu sa dá jednoducho zmeniť).

**Funkcia** **test** pracuje tak ako funkcia velkyTest, až na to, že vykoná testy len pre nezredukovaný diagram (spolu s jeho redukciou).

**Keď prebehnú všetky testy**, program vypíše počet nezhôd, počet testov, počet pôvodných a počet odstránených uzlov, a percentuálnu mieru redukcie uzlov.

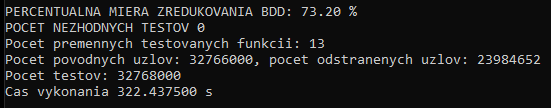
Výsledky testov

Funkcia **malyTest** (ukážka):

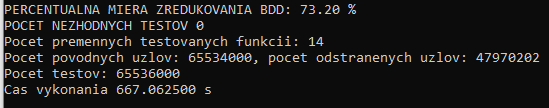


Funkcia **velkyTest** (nezredukovaný aj zredukovaný diagram, porovnanie výsledkov jednotlivo aj navzájom):

* 2000 funkcií, 13 premenných (5.3 min)

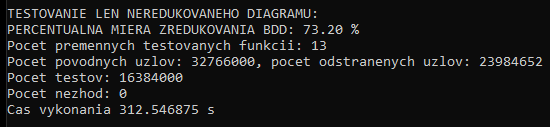


* 2000 funkcií, 14 premenných (11 min)

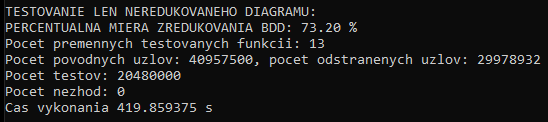


Funkcia **test**:

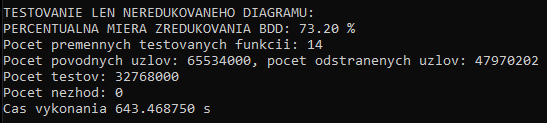
* 2000 funkcií, 13 premenných (5.2 min)



* 2500 funkcií, 13 premenných (6.9 min)



* 2000 funkcií, 14 premenných (10.7 min)



Priemerná komplexnosť je O(log n).

Priestorová zložitosť je priemerne konštantne lineárna.

