Marcel Valent
Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4
xvalentm@stuba.sk
30. Apríl 2021

ZADANIE 3 – Binárne rozhodovacie diagramy

Prednášajúci: Ing. Lukáš Kohútka, PhD.
Cvičiaci: Mgr. Martin Sabo, PhD.
Cvičenie: Utorok 16:00

Obsah

1. Zadanie

Vytvorte program, ktorý bude vedieť vytvoriť, redukovať a použiť dátovú štruktúru BDD (Binárny Rozhodovací Diagram) so zameraním na využitie pre reprezentáciu Booleovských funkcií.

Konkrétne implementujte 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**. Štruktúru BF si definujete sami – podstatné je, aby nejakým (vami vymysleným/zvoleným spôsobom) bolo možné použiť štruktúru BF na opis Booleovskej funkcie. Napríklad, BF môže opisovať Booleovskú funkciu ako pravdivostnú tabuľku, vektor, alebo výraz. Návratovou hodnotou funkcie **BDD_create** je ukazovateľ na zostavený binárny rozhodovací diagram, ktorý je reprezentovaný vlastnou štruktúrou BDD. Štruktúra BDD musí obsahovať minimálne tieto zložky: **počet premenných, veľkosť BDD (počet uzlov) a ukazovateľ na koreň (prvý uzol) BDD**.

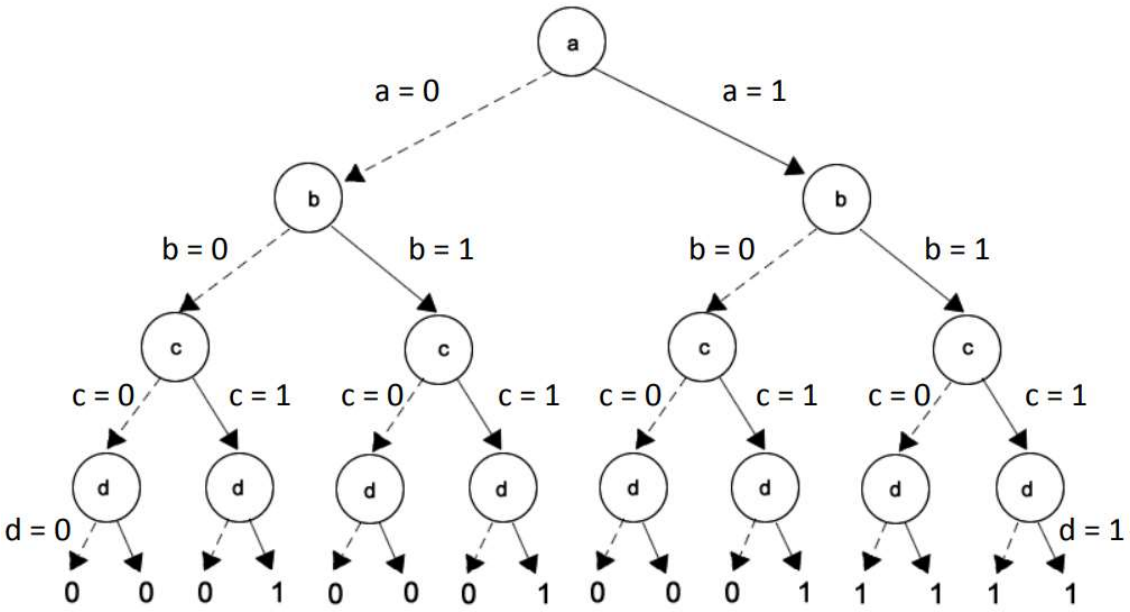
Funkcia **BDD_reduce** má slúžiť na redukciu existujúceho (zostaveného) binárneho rozhodovacieho diagramu. Aplikovaním tejto funkcie sa nesmie zmeniť Booleovská funkcia, ktorú BDD opisuje. Cieľom redukcie je iba zmenšiť BDD odstránením nepotrebných (redundantných) uzlov. Funkcia **BDD_reduce** dostane ako argument ukazovateľ na existujúci BDD (bdd), ktorý sa má redukovať. Redukcia BDD sa vykonáva priamo nad BDD, na ktorý ukazuje ukazovateľ bdd, a preto nie je potrebné vrátiť zredukovaný BDD návratovou hodnotou (na zredukovaný BDD bude totiž ukazovať pôvodný 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). Samozrejme, funkcia **BDD_reduce** má aktualizovať aj informáciu o počte uzlov v BDD.

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 nejakým (vami zvoleným) spôsobom konkrétnu kombináciu vstupných premenných Booleovskej funkcie. Napríklad, index poľa reprezentuje nejakú premennú a hodnota na tomto indexe reprezentuje hodnotu tejto premennej (t.j. pre premenné A, B, C a D, kedy A a C sú jednotky a B a D sú nuly, môže ísť napríklad o “1010“), môžete si však zvoliť iný spôsob.

2. Čo je to binárny rozhodovací diagram

BDD je dátová štruktúra ktorá má tvar, ktorý je totožný s binárnym stromom. Najčastejšie je používaný na reprezentáciu Booleovských funkcií.



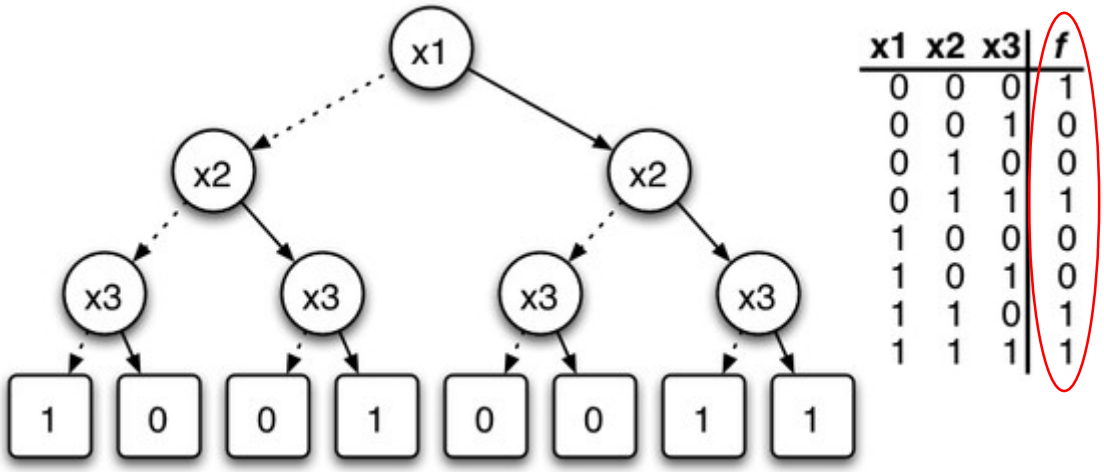
BDD

Booleovské funkcie môžu byť reprezentované nasledovnými spôsobmi:

- Výraz - DNF, KNF, rovnica
- Vektor výstupy pravdivostnej tabuľky
- Karnaughova mapa
- Pravdivostná tabuľka

2.1.Reprezentácia BDD pomocou vektora

V červenej bubline je zaznačený vektor, ktorý budeme používať na vytváranie BDD.



x1	x2	x3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Booleovská funkcia pomocou BDD

3. Implementácia

Štruktúry

Použité štruktúry v mojej implementácii.

```
typedef struct node {
    struct node* pravy;
    struct node* lavy;
    char data;
} NODE;
typedef struct bdd {
    int pocet_premennych;
    int uzly;
    NODE* hlavicka;
}BDD;
```

3.1.BDD* BDD_create(char* bfunkcia)

Táto funkcia má za úlohu vytvoriť BDD, kde ako parameter vstupuje do tejto funkcie v mojej implementácii náhodne vygenerovaný vektor Booleovskej funkcie. Za pomoci funkcie binary sa tento vstup rozloží na jednotlivé “0” a “1” a vytvorí sa BDD. Táto funkcia vracia už vytvorený BDD.

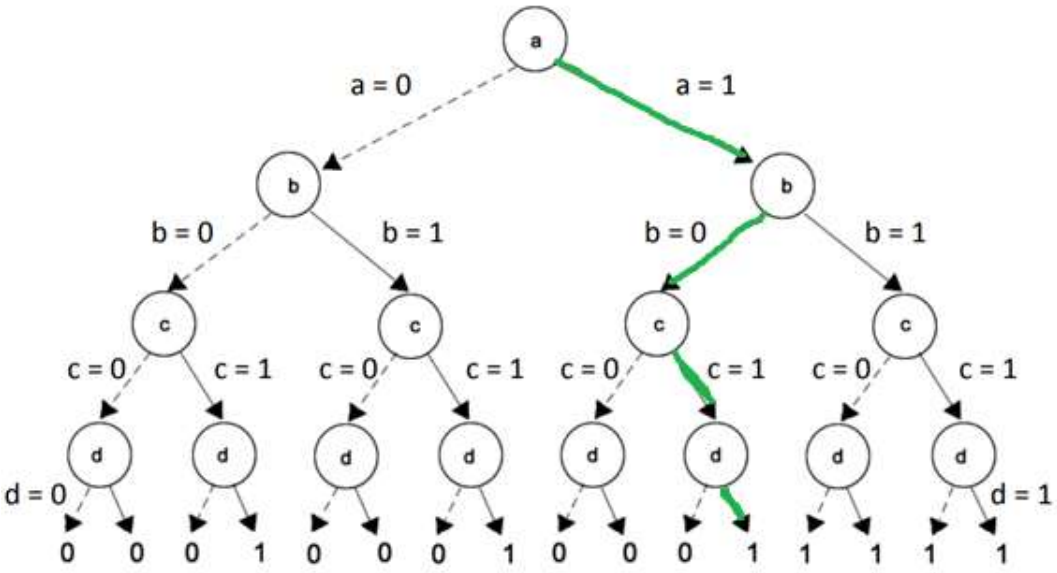
```
void binary(int cislo, int bites, char* pomocnepole) {
    for (int i = bites; i > 0; i--) {
        pomocnepole[i - 1] = cislo % 2;
        cislo /= 2;
    }
}
```

Pomocná funkcia binary

3.2.char BDD_use(BDD* bdd, char* vstupy)

Táto funkcia len prechádza už vytvoreným BDD, a na konci navráti hodnotu listu (“0” alebo “1”).

Na vstupe máme napríklad A=1, B=0, C=1, D=1 a cesta a výsledok je nasledovný:



BDD_USE(1011)

4. Testovanie

```
Priemerny cas BDD je: 0.001 sekund
Testovanie BBD o 10 premennych s poctom opakovani 2000 trvalo: 2.766 sekund.
```

```
Priemerny cas BDD je: 0.015 sekund
Testovanie BBD o 13 premennych s poctom opakovani 2000 trvalo: 30.050 sekund.
```

```
Priemerny cas BDD je: 0.171 sekund
Testovanie BBD o 15 premennych s poctom opakovani 500 trvalo: 85.283 sekund.
```

```
Testovana funkcia: 11001000001111111010
Vysledok funkcie je: 0
Testovanie tohto BDD stromu trvalo: 150.990 sekund

Priemerny cas BDD je: 150.990 sekund
Testovanie BBD o 20 premennych s poctom opakovani 1 trvalo: 150.990 sekund.
```

Postupne som zvyšoval počet premenných a znižoval počet opakovaní z dôvodu časovej náročnosti programu, smerodajné pre nás sú ale priemerné časy vytvorenia a použitia BDD. Tu si môžeme všimnúť, že časová náročnosť podľa očakávania stúpala exponenciálne s počtom premenných, a to z dôvodu, že vektor Booleovskej funkcie

má tvar **2 na n-tú**, kde n je počet premenných. Taktiež som skúšal aj test s 25 premennými, ktorý som bol ale nútený po troch hodinách vypnúť. Ale z predchádzajúcich testov je zrejmé, že tento test by exponenciálny nárast času len potvrdil.