Dokumentácia – Zadanie 2

Vypracoval: Filip Paučo

AIS ID: 116270

Obsah

1. Binárne rozhodovacie diagramy

1.1. Teória

Binárne rozhodovacie diagramy sú spôsob akým vieme reprezentovať booleovské funkcie. Tento diagram je acyklický graf s uzlami, ktoré sú vertikálne prepojené hranami. Jednou hranou z vrchu a dvoma zo spodu. Tam nastáva rozhodovanie jedna hrana je reprezentovaná hodnotou 0 a druhá hodnotou 1 pre zvolenú premennú. Každý uzol obsahuje String s funkciou a dva ukazovatele na pravý a ľavý uzol. Na samotnom vrchu v uzle je položená pôvodná funkcia, ktorá je určená na rozkladanie.

V jednotlivých funkciách využívame len 3 základné hradlá – násobenie
– sčítanie
– negácia

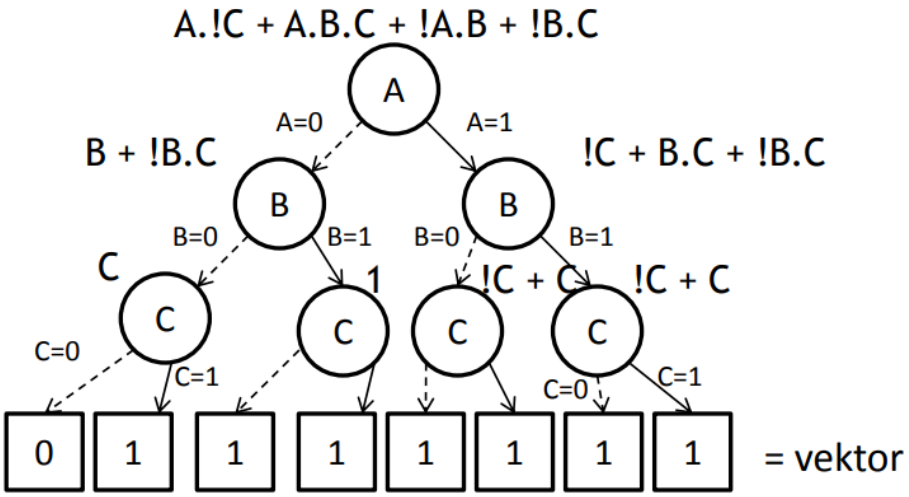
Pomocou týchto operátorov vieme upraviť všetky booleovské funkcie. Ďalšie hradlá ako nand, nor, xor, xnor sú len kombinácie pôvodných premenných, sčítaní, násobení a negácí.

1.2. Shannonova dekompozícia

Shannonova dekompozícia je spôsob vytvárania diagramu, kde dosadzujeme do ľavého uzlu 0 a do pravého 1. Takto pokračuje až pokým nedostaneme v uzle hodnotu 0 alebo 1. Získanie funkcie prvého ľavého uzlu: (0.!C + 0.B.C + 1.B + !B.C) -> (B + !B.C). Získanie hodnoty prvého pravého uzlu: (1.!C + 1.B.C + 0.B + !B.C) -> (!C + B.C + !B.C). Celkový počet uzlov vypočítame vzorcom $2^{\text{počet premenných} + 1} - 1$ (v tomto prípade je to 15). Jeden z pojmov, ktorý sa používa v tejto problematika je **vektor**. Vektor je výsledná hodnota nejakej cesty v grafe. Hodnota vektora je buď 0 alebo 1.

Obr. 1- príklad dekompozície v poradí (A, B, C)

(zdroj: DSA prednáška č.7 - Lukáš Kohútka)



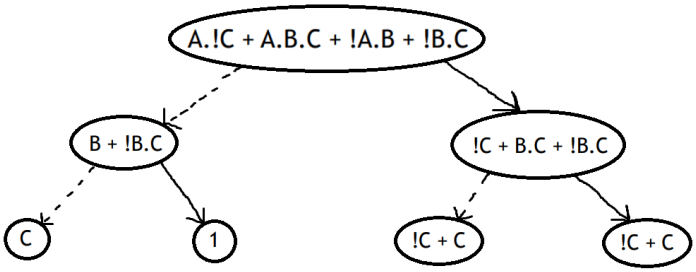
1.3. Redukcia

Veľmi podstatnú časť tohto zadania je vytvorenie redukcie štruktúry tak aby bola čo najefektívnejšia. Ja som to riešil v priebehu tvorby stromu. Vždy predtým než vložím uzol na ľavú alebo pravú stranu, prejdem si štruktúru a vyhľadám či daný prvok už v strome existuje. Pokiaľ áno príslušný ukazovateľ (ľavý alebo pravý) napojím na tento uzol. V prípade, že tento uzol v strome ešte nie je, uzol, ktorý porovnávam nastavím ako ukazovateľ na príslušnej strane. Pri vytváraní diagramu taktiež sledujeme aj mieru redukcie. Toto číslo získame zo vzorca počet zjednodušených uzlov / počet všetkých uzlov bez redukcie.

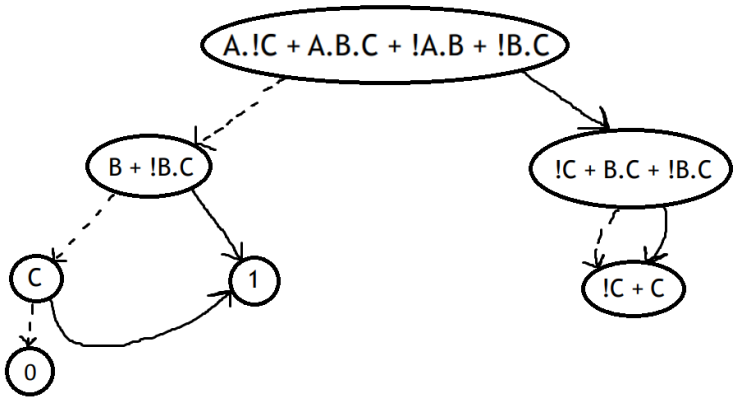
Obr. 2, 3, 4 - príklad redukcie v danej funkcii

(zdroj: DSA prednáška č.7 - Lukáš Kohútka)

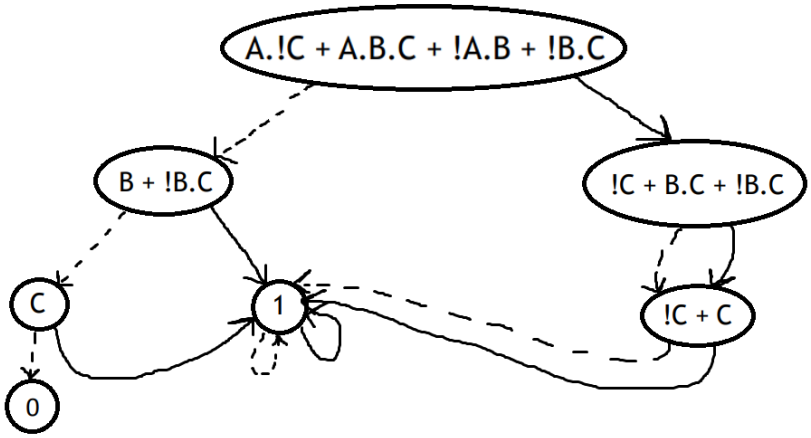
2.



3.



4.



Obr. 5 - Vstup do funkcie BDD_create

```
A - Nenegovana hodnota || a - Negovana hodnota || Rozmedzie premennych A az M vratane
Zadaj funkciu vo formate napr: A*b+C
A*c+A*B*C+a*B+b*C
Zadaj poradie premennych (je potrebne zadat vsetky premennne, ktore boli zadane vyssie)
ABC
```

Obr. 6 - výstup funkcie, uzly, ktoré sa nachádzajú v diagrame

```
    -----************-----
A*c+A*B*C+a*B+b*C
B+b*C
C
0
1
c+B*C+b*C
c+C
    -----************-----
```

Obr. 7 – ďalší výstup funkcie, informácie o vytvorenom diagrame

```
    -----***********-----
Celkovy pocet uzlov: 15
Zjednodusil som: 8 uzlov
Pocet (jedinecnych) uzlov po redukcii: 7
Miera redukcie je priblizne: 53.0 %
    -----***********-----
```

1.4. BDD_create a BDD_use

Funkcia **BDD_create** slúži na vytvorenie redukovaného diagramu s danej funkcie. Na vstupe dostáva funkciu a poradie premenných podľa ktorých bude nasledovať rozklad. V rámci tejto funkcie som využil množstvo funkcií na úpravu stringov, prehľadávanie diagramu a aj samotný výpis informácií o diagrame ako na obrázku 7.

Obr.8 – Kontrola či už uzol existuje, alebo je potrebné ho vytvoriť.

```
String crd = cr.toUpperCase();
String cbd = cr.toLowerCase();
if(!node.funkcia.equals("0") && !node.funkcia.equals("1") && (node.funkcia.contains(crd) || node.funkcia.contains(cbd))) {
    node.left = search(temp,anotherN.funkcia);
    if(node.left  != null){
        pocet++;
        anotherN = null;
    }
    else{
        node.left= anotherN;
    }
```

Na obrázku č.8 vytváram s každého uzlu ľavý a pravý, iba v prípade že aktuálny uzol neobsahuje iba 0, iba 1, alebo ak neobsahuje danú premennú alebo premennú v negovanej podobe. V takom prípade len vyhľadám iba duplikát a pri nájdení zvýšim premennú počet o 1. Ak však vstupujem do prvého if – bloku tak opäť hľadám duplikát, ak ho nájdem, zvýšim premennú počet o 1 a napojím uzol na nájdený a pôvodný zmažem. Ak duplikát nenájdem, napojím uzol na upravovaný uzol. Tento postup opakujem aj pre pravú stranu.

Funkcia **BDD_use** slúži na vyhľadanie cesty vo vytvorenom diagrame podľa hodnôt premenných zadaných v poradí. Na vstupe je diagram a string hodnôt premenných (0,1). Návratom funkcie je znak 0 alebo 1. To nám udáva pravdivostnú hodnotu danej funkcie pre zadané hodnoty premenných.

Obr.9 – funkcia BDD_use

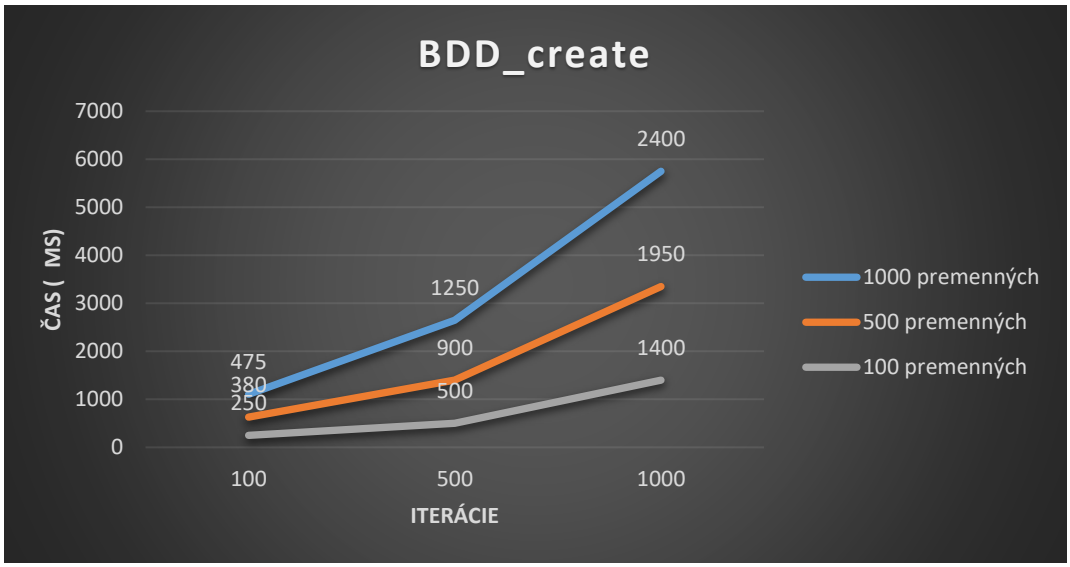
```
public String Bdd_use(Node root,char c[]){
    int i;

    for(i=0;i<c.length;i++){
        if(root.right == null || root.left == null)
            break;
        if(c[i] == '1')
            root = root.right;
        if(c[i] == '0')
            root = root.left;
    }
    String cd = root.funkcia;
    return cd;
}
```

2. Testovanie

2.1 BDD_create

Moje testy závisia od dvoch premenných, a to počet premenných booleovskej funkcie(môžu sa opakovať, nemusia byť nutne jedinečné) a počet iterácií, alebo diagramov, ktoré som vytvoril. Daný počet náhodných premenných som vytváral pomocou funkcie, ktorá náhodne zvolila, ako premenné, tak aj operácie medzi nimi. Tieto merania som aj zmeral časom, ktorý je v grafoch uvedený v milisekundách. Všetky ďalšie potrebné údaje by mali byť jasne čitateľné z tabuliek a grafov.

premenné/iterácie	100 i	500 i	1000 i	redukcia
100 p	250	500	1400	99 %
500 p	380	900	1950	99 % +
1000 p	475	1250	2400	99 % +



2.2 BDD_use

premenné/iterácie	100 000 i	500 000 i	1 000 000 i
100 p	5	8	8
500 p	16	18	20
1000 p	18	23	23