9/5/2020

Binárne rozhodovacie diagramy

DSA – Zadanie 3

Samuel Hetteš
ID: 110968
STU FIIT 2020/2021

OBSAH

ZADANIE – Binárne rozhodovacie diagramy

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);

Samozrejme môžete implementovať aj ďalšie funkcie, ktoré Vám budú nejakým spôsobom pomáhať v implementácii vyššie spomenutých funkcii, nesmiete však použiť existujúce funkcie na prácu s binárnymi rozhodovacími diagramami.

IMPLEMENTÁCIA

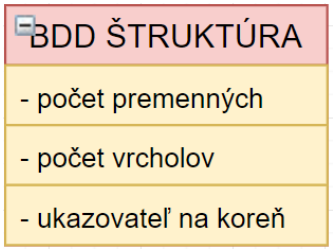
- **metóda:** boolovská funkcia opísaná **vektorom**
- v mojej implementácii som pre rýchlejšie porovnávanie dát použil **prevzatú hashovaciu funkciu djb2**
- zdroj: http://www.cse.yorku.ca/~oz/hash.html
- v testeri som pre usporiadanie poľa použil prevzatú funkciu quick sort zo zdroja: https://www.geeksforgeeks.org/quick-sort/
- program obsahuje jeden hlavičkový súbor bdd.h, zdrojový súbor bdd.c a tester.c, ktorý obsahuje funkciu main a tester

ŠTRUKTÚRY

- implementované sú nasledovné štruktúry:

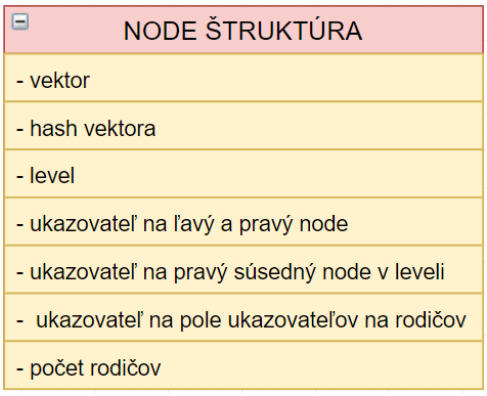
ŠTRUKTÚRA BDD

- obsahuje **iba požadované zložky**, ktoré boli uvedené v zadaní



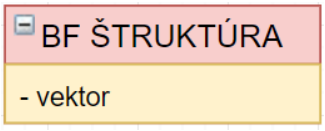
ŠTRUKTÚRA UZLA

- pre rýchlejšiu prácu pri porovnávaní reťazcov – **hash vektora**
- pre rýchlejšiu manipuláciu pri redukovaní – **pravý sused, pole rodičov**
- pre rýchlejšiu prácu pri používaní diagramu – **level**
- **najnižší node má level 0**
- **koreň má level rovný počtu premenných**



ŠTRUKTÚRA BOOLOVSKEJ FUNKCIE

- opísaná pomocou **vektora** – reťazec



OPIS FUNKCIÍ

BDD *BDD_CREATE(BF *bfunction)

- **metóda:** zostrojenie stromu zhora nadol rekurzívnym spôsobom

Riešenie:

- zavolá sa funkcia **check_function(bfunction)**, ktorá **overí boolovskú funkciu** a **vráti počet premenných**
- **alokuje** sa miesto pre **BDD štruktúru** a nastavia parametre
- zavolá sa funkcia **create_diagram(BDDNODE *root, char *vector, int level)**, ktorá **rekurzívnym spôsobom** vytvorí **uplný strom**, zabezpečí zahashovanie a delenie vektora v jednom cykle a nastaví prvého rodiča
- **hashovacia funkcia: djb2**
    - zdroj: http://www.cse.yorku.ca/~oz/hash.html

```
unsigned long hash = 5381;
int c;
while(c = *vector++)
    hash = ((hash << 5) + hash) + c;    // hash * 33 + c
```

- pre vytvorenie nového uzla slúži funkcia **create_node(char *vector, int level, unsigned long hash)**, ktorá **alokuje** miesto pre **nový uzol,** nastaví parametre
- ak bolo vytvorenie stromu úspešné, zavolá sa funkcia **connect_neighbours(BDDNODE *root)**, ktorá **rekurzívnym spôsobom jednosmerne pospája uzly v leveli** počnúc od ľavého uzla
- funkcia vráti ukazovateľ na novo-vytvorený BDD diagram

int BDD_REDUCE(BDD *bdd)

- **metóda:** redukcia zdola nahor iteratívnym spôsobom

Riešenie:

- **overia sa** vstupné **parametre** a **nájde ľavý dolný uzol**
- začnú sa vykonávať viaceré **vnorené cykly**, ktoré **zabezpečujú redukciu smerom zdola nahor**
- cyklus nad ľavým uzlom, ktorý sa posúva vždy o level vyššie:
    - porovnávaný uzol = ľavý uzol
    - cyklus nad porovnávaným uzlom, ktorý sa posúva po susedoch doprava

REDUCE TYP I

- cyklus, ktorý porovná porovnávaný uzol so všetkými uzlami vpravo od neho
    - najskôr sa porovnajú hashe, ak sa zhodujú porovnajú sa dané vektory
    - v prípade zhody sa odpojí uzol v leveli a zavolá funkcia **reduce_I**, ktorá **prenastaví** rodičom nadbytočného uzlu **potomkov**, porovnávanému uzlu **pridá rodiča** (**ak sa nezhoduje už s pridanými**) - realloc a uvoľní miesto

REDUCE TYP S

- po odstránení duplikátov sa porovnajú ukazovatele na potomkov porovnávaného uzla
- v prípade zhody sa odpojí uzol v leveli a zavolá funkcia **reduce_S**, ktorá **skontroluje či** nadbytočný uzol **nie je koreň**, **nastaví rodičom** nadbytočného uzlu **nových potomkov** – potomka nadbytočného uzlu a jemu naopak **nových rodičov** – realloc, uvoľní miesto

- aktualizuje sa počet uzlov v bdd
- funkcia vráti počet odstránených uzlov

- ak by vstupné parametre boli nesprávne alebo došlo ku chybe napr. pri realokácii miesta pre rodičov, funkcia vráti -1

char BDD_use(BDD *bdd, char *input)

Riešenie:

- overia sa vstupné parametre, overí sa dĺžku vstupu
- v prípade chyby funkcia vráti -1
- začne sa cyklus, ktorý prechádza uzlami až k listu:
    - ak sa level uzla nezhoduje so zostávajúcou dĺžkou vstupu preskoč daný index
    - inak prejdi doľava ak znak je '1', doprava ak '0', inak vráť -1 (nekorektný vstup)
    - dekrementuj zostávajúcu dĺžku vstupu
- vráť hodnotu vektora v liste (symbol)

BDD_free(BDD *bdd)

- funkcia slúži pre uvoľnenie pamäti alokovanej pre BDD
- zavolá sa funkcia free_diagram, ktorá rekurzívnym spôsobom uvoľní daný strom – uvoľnovaný uzol sa vždy odpojí od svojich rodičov
- po skončení funkcie free_diagram sa uvoľní pamäť, ktorá bola alokovanú pre samotnú štruktúru BDD

ČASOVÁ A PRIESTOROVÁ ZLOŽITOSŤ

BDD_CREATE

- časová zložitosť:
    - n – dĺžka vstupného vektora

  - (2n - 1) – počet uzlov
  - (n.log$_2$n) – počet cyklov celkovo pri hashovaní a delení vektorov
  - (2n – 1 + n.log$_2$n) = (n.(2 + log$_2$n) - 1) -> O(n.logn)
- priestorová zložitosť:
  - n – dĺžka vstupného vektora
  - (2n - 1) – počet štruktúr
  - (n.log$_2$n) – miesto alokované pre uloženie vektorov
  - (2n – 1 + n.log$_2$n) = (n.(2 + log$_2$n) - 1) -> O(n.logn)

BDD_REDUCE

- časová zložitosť: O(n!) – n je dĺžka vektora boolovskej funkcie
- priestorová zložitosť: O(n) – n je počet unikátnych rodičov odstránených uzlov

BDD_USE

- časová zložitosť:
  - n – dĺžka vektora boolovskej funkcie
  - (log$_2$n) – počet cyklov pre zistenie výstupu
  - O(logn)
- priestorová zložitosť: O(1)

TESTOVANIE

- testovanie je implementované v súbori tester.c
- v main funkcii je možné nastaviť si počet premenných danej boolovskej funkcie a počet funkcií, ktoré sa majú vygenerovať

TESTER

- **vstupné parametre:** počet premenných, počet generovaných funkcií
- celé testovanie spočíva na **cykle**, ktorý **iteruje na základe počtu generovaných funkcií**
    - v každom behu sa **vygeneruje náhodný vektor** a **zavolajú funkcie BDD_create a BDD_reduce**
    - začne sa **cyklus, ktorý testuje** daný **diagram** pre **každý** možný **vstup**, teda pre vstupy **<0; 2^premenné – 1>**
        - daný **vstup sa prevedie z desiatkového čísla na binárny reťazec** a **zavolá sa** funkcia **BDD_use**
        - **kontrolovanie funkcie BDD_use:**
        - **návratová hodnota funkcie sa porovná s hodnotou vektora na idexe**, ktorý predstavuje daný binárny reťazec - vstup (priamy prístup)
        - ak sa nezhodujú vypíše sa chyba
    - **zavolá sa** funkcia **BDD_free**, ktorá uvoľní pamäť
    - počas behu je zaznamenávaný čas trvania jednotlivých funkcií, ako aj pomer zredukovania stromu v poli
- na záver testovania sa vypíše správa o nastavených vstupných parametroch, priemerná percentuálna miera zredukovania, medián zredukovania, počet diagramov v jednotlivých rozmedziach miery zredukovania po pol-desatinách percenta, priemerný čas potrebný na vykonanie jednotlivých operácií a celkový čas

TEST Č. 1

- počet premenných: 10
- počet generovaných funkcií: 2000
- výsledky:

```
------------------------------------------------------------
--------------------- TEST RESULTS ---------------------
------------------------------------------------------------
REDUCTION RATE:
-> AVERAGE: 88.44%
-> MEDIAN: 88.42%
-> [87.95% - 88.00%]: 31
-> [88.00% - 88.05%]: 31
-> [88.10% - 88.15%]: 62
-> [88.15% - 88.20%]: 94
-> [88.20% - 88.25%]: 124
-> [88.25% - 88.30%]: 155
-> [88.30% - 88.35%]: 284
-> [88.35% - 88.40%]: 157
-> [88.40% - 88.45%]: 126
-> [88.45% - 88.50%]: 188
-> [88.50% - 88.55%]: 124
-> [88.55% - 88.60%]: 186
-> [88.60% - 88.65%]: 157
-> [88.65% - 88.70%]: 63
-> [88.70% - 88.75%]: 31
-> [88.75% - 88.80%]: 63
-> [88.80% - 88.85%]: 62
-> [88.90% - 88.95%]: 31
-> [88.95% - 89.00%]: 31

AVERAGE TIME:
-> CREATE: 0.000460s
-> REDUCE: 0.000290s
-> USE: 0.000000s

TOTAL TIME:
-> CREATE: 0.920000s
-> REDUCE: 0.580000s
-> USE: 0.170000s
-> ALL: 1.670000s
```

TEST Č. 2

- počet premenných: 13
- počet generovaných funkcií: 2000
- výsledky:

```
------------------------------------------------------------
--------------------- TEST RESULTS ---------------------
------------------------------------------------------------
REDUCTION RATE:
-> AVERAGE: 92.24%
-> MEDIAN: 92.24%
-> [92.20% - 92.25%]: 1500
-> [92.25% - 92.30%]: 500

AVERAGE TIME:
-> CREATE: 0.003095s
-> REDUCE: 0.004398s
-> USE: 0.000000s

TOTAL TIME:
-> CREATE: 6.190000s
-> REDUCE: 8.796000s
-> USE: 1.392000s
-> ALL: 16.378000s
```

TEST Č. 3

- počet premenných: 15
- počet generovaných funkcií: 2000
- výsledky:

```
------------------------------------------------------------
--------------------- TEST RESULTS ---------------------
------------------------------------------------------------
REDUCTION RATE:
-> AVERAGE: 93.40%
-> MEDIAN: 93.40%
-> [93.35% - 93.40%]: 1000
-> [93.40% - 93.45%]: 1000

AVERAGE TIME:
-> CREATE: 0.014130s
-> REDUCE: 0.048785s
-> USE: 0.000000s

TOTAL TIME:
-> CREATE: 28.261000s
-> REDUCE: 97.571000s
-> USE: 6.137000s
-> ALL: 131.969000s
```

TEST Č. 4

- počet premenných: 17
- počet generovaných funkcií: 2000
- výsledky:

```
------------------------------------------------------------
--------------------- TEST RESULTS ---------------------
------------------------------------------------------------
REDUCTION RATE:
-> AVERAGE: 93.81%
-> MEDIAN: 93.81%
-> [93.80% - 93.85%]: 2000

AVERAGE TIME:
-> CREATE: 0.063555s
-> REDUCE: 0.782745s
-> USE: 0.000000s

TOTAL TIME:
-> CREATE: 127.109000s
-> REDUCE: 1565.489000s
-> USE: 29.525000s
-> ALL: 1722.123000s
```

VÝSLEDKY TESTOVANIA:

- na základe vykonaných testov som prišiel k nasledovným zisteniam:
- **hashovanie vektorov, prepojenie susedov a pole rodičov:** pri malom počte premenných trvá samotné vytvorenie stromu dlhšie ako redukcia, no pri väčších počtoch premenných nám tieto operácie výrazne zvýšia rýchlosť a efektivitu redukcie
- **pridávanie iba rodičov, ktorých vektor je unikátny:** pri redukcii sa rodič uvoľňovaného uzlu pridá iba v prípade, že je unikátny – ak nie je nemusíme ho pridávať, pretože bude ajtak odstránený – týmto spôsobom sa výrazne zvýšila efektivita redukcie, keďže nemusíme upravovať až také veľké množstvo ukazovateľov
- **uchovávanie levelu v danom uzli:** týmto spôsobom sa výrazne zrýchlila efektivita používania daného diagramu – nemusíme kontrolovať dĺžku vektora v uzle, kontrolujeme level
- **s väčším počtom premenných rastie miera redukcie** diagramu a **percentuálny rozptyl miery redukcie sa výrazne zmenšuje**
- **pridanie** čo i len **jednej premennej, rapídne ovplyvní čas** potrebný na vytvorenie a redukciu diagramu
- **používanie diagramu je veľmi rýchle** – priemerný čas ani nevyčíslilo

ZÁVER

Mojim cieľom bolo vytvoriť implementáciu, ktorá bude efektívna najmä pri veľkom počte premenných. Tento zámer sa mi podaril docieliť prepojením uzlov v leveli, uchovávaním rodičov, hashovaním vektorov, pridávaním iba unikátnych rodičov pri redukcii a uchovávaním levelu v danom uzli.

Pre menší počet premenných môže byť táto implementácia nie až taká efektívna, no zadanie bolo zamerané na veľký počet premenných, čo sa mi myslím podarilo.