Lukáš Michalík, ID:103052
Pondelok 09:00

Slovenská Technická Univerzita v Bratislave
Fakulta informatiky a informačných technológií

Dátové štruktúry a algoritmy

Zadanie 3 – Binárne rozhodovacie diagramy

Popis funkcií

BDD_create

Funkcia **BDD_create** slúži na zostavenie úplného binárneho rozhodovacieho diagramu, ktorý má reprezentuje zadanú Booleovskú funkciu vo forme **BF štruktúry**, na ktorú ukazuje samotný ukazovateľ, ktorý je zadaný ako argument funkcie **BDD_create**. Samotný BDD diagram je reprezentovaný štruktúrou **BDD** a jednotlivé prvky diagramu (uzly) sú reprezentované štruktúrou **ELEMENT**.

Postup pri vytváraní BDD diagramu:

Najskôr sa vytvorí (alokuje) samotná štruktúra **BDD** a v nej sa vytvorí prvý uzol. Následne sa rekurzívnym spôsobom, pomocou funkcií ***createNewLeftElement(...)*** a ***createNewRightElement(...)*** vytvárajú ďalšie poduzly so zmenšenou (upravenou) Booleovskou funkciou. Nakoniec funkcia **BDD_create** vráti ukazovateľ na úspešne vytvorený BDD diagram.

BF štruktúra sa skladá z:

```
typedef struct bf {
    char *vector;
    int vectorSize;
} BF;
```

- vektor (string), v ktorý reprezentuje samotnú Booleovskú funkciu
- veľkosť vektora (dĺžka)

BDD štruktúra sa skladá z:

```
typedef struct bdd {
    int variableCount;
    int elementCount;
    struct element *firstElement;
} BDD;
```

- počet premenných, Booleovskej funkcie
- počet všetkých uzlov v diagrame
- ukazovateľ na prvý prvok (uzol) v diagrame

ELEMENT štruktúra sa skladá z:

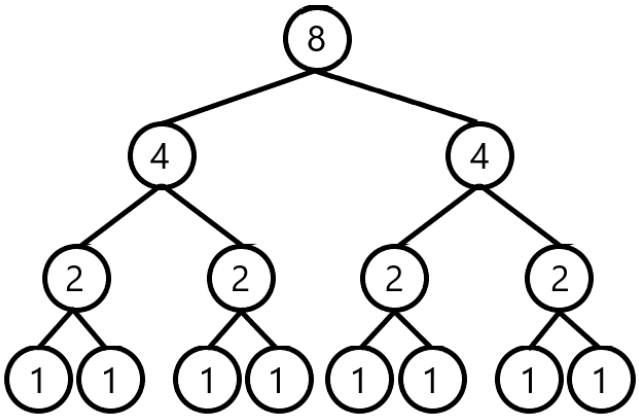
```
typedef struct element {
    int stage;
    char side;
    bool originalLeft;
    bool originalRight;
    BF booleanFunction;
    struct element *parentElement;
    struct element *neighborElement;
    struct element *leftElement;
    struct element *rightElement;
} ELEMENT;
```

- hĺbka uzla (stage)
- strany (ľavá/pravá)
- kontrolné premenné na kontrolu, či L/R prvok bol odstránený
- štruktúra na opis Booleovskej funkcie (postupne redukovaná)
- ukazovateľ na rodičovský uzol
- ukazovateľ na susedný uzol v tej istej hĺbke
- ukazovatele na L/R poduzol

Funkcia **createNewLeftElement(...) / createNewRightElement(...)**

```c
ELEMENT *createNewLeftElement(BDD *newBDD, ELEMENT *parent, int size, char *vector, int stage, ELEMENT **neighborStack) {

    int newSize = size/2;

    if (size == 1)
        return NULL;

    char *newVector = malloc(newSize * sizeof(char));

    //prekopirovanie vektora
    for (int i = 0; i < newSize; i++) {
        newVector[i] = vector[i];
    }

    ELEMENT *newElement = malloc(sizeof(ELEMENT));

    newElement->stage = stage;
    newElement->side = 'l';
    newElement->booleanFunction.vector = newVector;
    newElement->booleanFunction.vectorSize = newSize;

    newElement->originalLeft = true;
    newElement->originalRight = true;

    newElement->parentElement = parent;
    newElement->leftElement = createNewLeftElement(newBDD, newElement, newSize, newVector, stage+1, neighborStack);
    newElement->rightElement = createNewRightElement(newBDD, newElement, newSize, newVector, stage+1, neighborStack);
    newElement->neighborElement = NULL;

    if (neighborStack[stage] == NULL)
        neighborStack[stage] = newElement;
    else {
        neighborStack[stage]->neighborElement = newElement;
        neighborStack[stage] = newElement;
    }

    //printf("IDEM\n");
    newBDD->elementCount++;

    return newElement;
}
```

Grafická reprezentácia zmenšovania (úpravy) veľkosti Booleovskej funkcie v jednotlivých uzloch:



BDD_use

Funkcia **BDD_use** má slúži na použitie BDD diagramu pre zadanú kombináciu vstupných premenných Booleovskej funkcie a zistenie výsledku Booleovskej funkcie pre túto kombináciu vstupných premenných.

Argumentami funkcie **BDD_use** je ukazovateľ s názvom ***bdd*** ukazujúci na BDD diagram (ktorý sa má použiť) a ukazovateľ s názvom ***input*** ukazujúci na začiatok poľa charov, v ktorom je uložená konkrétna kombinácia vstupných premenných.

Funkcia **BDD_use** prechádza BDD diagram smerom od prvého uzlu po posledný (s veľkosťou 1) takou cestou, ktorú určuje práve zadaná kombinácia vstupných premenných.

Pri vstupe = 1, sa v diagrame posúva doprava a pri vstupe = 0, sa v diagrame posúva doľava. Napr. pre kombináciu vstupných premenných „011“ sa v diagrame uskutoční posun LRR.

Návratová hodnota funkcie **BDD_use** je char, ktorý reprezentuje výsledok Booleovskej funkcie – je to buď ‘1’ alebo ‘0’. V prípade chyby je to hodnota 3 alebo 4 (podľa typu chyby).

```
char BDD_use(BDD *bdd, char *input) {

    ELEMENT *actualElement;
    int inputSize;

    if (bdd == NULL)
        return 3;

    actualElement = bdd->firstElement;
    inputSize = bdd->variableCount;

    for (int i = 0; i < inputSize; i++) {
        if (input[i] == 0)
            actualElement = actualElement->leftElement;
        if (input[i] == 1)
            actualElement = actualElement->rightElement;
    }

    if (actualElement->booleanFunction.vectorSize > 1)
        return 4;
    else
        return actualElement->booleanFunction.vector[0];
}
```

BDD_reduce

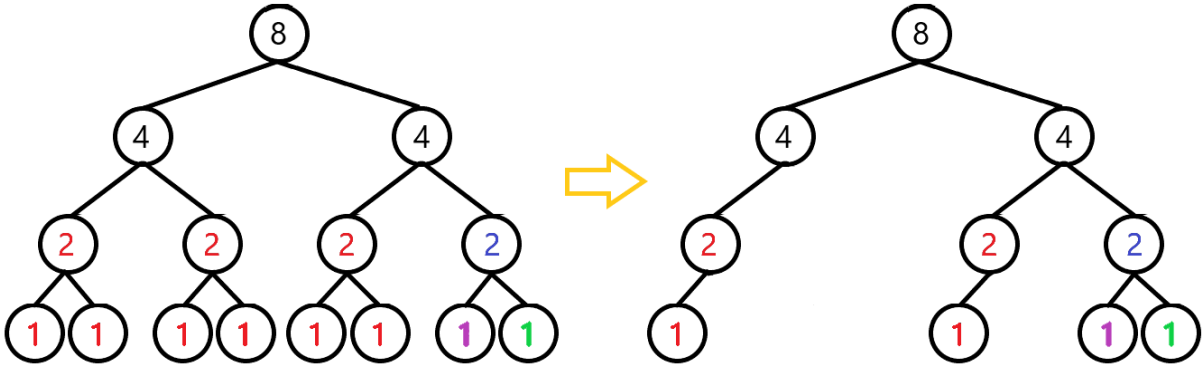
Funkcia **BDD_reduce** slúži na redukciu existujúceho BDD diagramu. Aplikovaním tejto funkcie sa však nemení Booleovská funkcia, ktorú BDD opisuje. Cieľom funkcie **BDD_reduce** je iba zmenšiť BDD diagram odstránením nepotrebných (redundantných) uzlov.

Funkcia dostane ako argument ukazovateľ na existujúci BDD diagram, ktorý sa má zredukovať. Redukcia BDD sa vykonáva priamo nad BDD, na ktorý ukazuje ukazovateľ bdd v argumente funkcie.

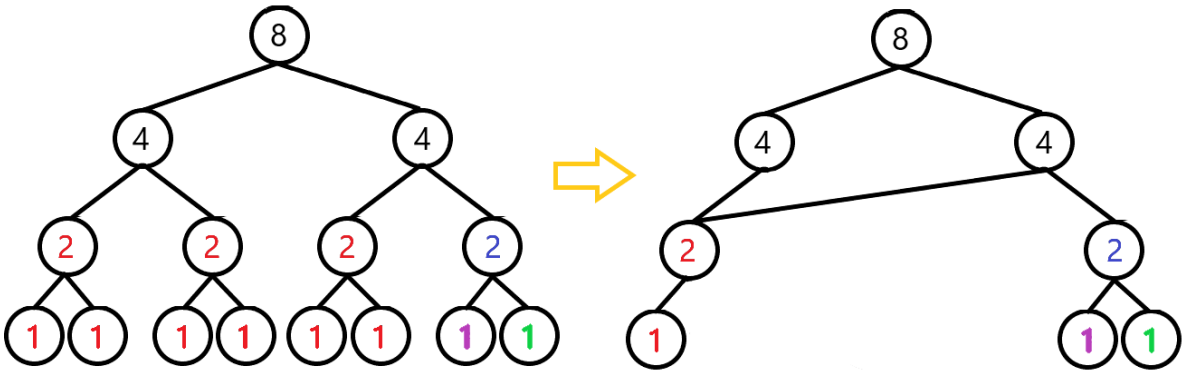
Návratovou hodnotou funkcie **BDD_reduce** je číslo typu *int*, ktoré vyjadruje počet odstránených uzlov. Ak je toto číslo záporné, vyjadruje nejakú chybu. Funkcia **BDD_reduce** taktiež aktualizuje aj informáciu o počte uzlov v BDD po zredukovaní.

Moja implementácia **BDD_reduce** neodstraňuje všetky redundantné uzly, ale iba poduzly v rámci daného uzlu.

Moja implementácia (červená farba = redundantné uzly):



Ideálna implementácia (červená farba = redundantné uzly):



Avšak pokúsil som sa aj o **ideálnu implementáciu**, ale neúspešne. Nefungovala konštantne a nepodarilo sa mi v nej nájsť chybu. Je súčasťou odovzdaného kódu. Stačí len odkomentovať zakomentovaný a zakomentovať ten nad ním. Tj. použiť namiesto funkcie ***checkElements(...)*** funkciu ***checkElementsNew(...),*** ale v odovzdanom kóde ju **nevyužívam**.

```
int BDD_reduce(BDD *bdd) {

    ELEMENT *actualElement;
    int deletedElements;

    if (bdd == NULL) {
        return -1;
    }

    actualElement = bdd->firstElement;
    deletedElements = checkElements(actualElement);
    //deletedElements = checkElementsNew(actualElement);
    bdd->elementCount -= deletedElements;

    return deletedElements;
}
```

Testovanie

Na testovanie som si vytvoril funkciu **BDD_test,** ktorá ako argumenty prijíma počet premenných a počet BDD diagramov. Vo tejto funkcii sa spustí cyklus, v ktorom sa postupne vytvárajú a testujú BDD diagramy pre zadaný **počet premenných** pre **rôzne Booleovské funkcie**, ktoré sú **náhodne generované** tiež podľa zadaného počtu premenných.

1. Najskôr sa vytvorí náhodná Booleovská funkcia.

Na náhodné generovanie Booleovských funkcií slúžia funkcie:

```
int sizeOfInput(int variableCount);
char *createInput(int inputSize);
BF *createBooleanFunction(int variableCount);
```

- výpočet veľkosti vektora (2^x)
- generovanie náhodného vektora z 0 a 1 danej veľkosti
- alokovanie a vytvorenie štruktúry BF podľa počtu premenných

2. Ďalej sa vytvorí BDD diagram pre vygenerovanú Booleovskú funkciu.

3. Spustí sa cyklus, v ktorom sa vytvárajú všetky možné kombinácie vstupov z 0 a 1 pre daný počet premenných. Následne sa každá vygenerovaná kombinácia vstupu pošle ako argument pre **BDD_use** a vyhodnotí sa výsledok. Každý výsledok sa priebežne ukladá do poľa s výsledkami ***resultsTable***, pomocou ktorého sa budú výsledky porovnávať po vykonaní operácie **BDD_reduce**.

```
vectorSize = bdd->firstElement->booleanFunction.vectorSize;
resultsTable = calloc(vectorSize, sizeof(char));

for (int j = 0; j < vectorSize; j++) {
    input = decToBin(bdd->variableCount, j);
    result = BDD_use(bdd, input);
    resultsTable[j] = result;
    free(input);
}
```

4. Zavolá sa funkcia **BDD_reduce**, ktorá daný BDD diagram zredukuje.

5. Znovu sa spustí sa cyklus, v ktorom sa vytvárajú všetky možné kombinácie vstupov z 0 a 1 pre daný počet premenných ako v bode 4. Teraz sa však navyše porovnávajú výsledky z **BDD_use** pred použitím **BDD_reduce**, ktoré sú uložené v poli ***resultsTable,*** s výsledkami, ktoré vracia **BDD_use** po zredukovaní diagramu.

6. Na koniec sa celý BDD diagram uvoľní funkciou **BDD_free**.

7. Tento proces sa opakuje pre x-krát podľa zadaného počtu diagramov ako argument funkcie **BDD_test**.

Výsledky testovania

```
PS C:\Users\Dzinak\Documents\FIIT\4. Semester\DSA\Zadania\zadanie3> cd "c:\Users
Cely test pre pocet premennych (13) a 2000 BDD diagramov trval 9.462000 sekund
Priemerna miera redukovanosti diagramov bola 22.37%
```

```
PS C:\Users\Dzinak\Documents\FIIT\4. Semester\DSA\Zadania\zadanie3> cd "c:\Users
Cely test pre pocet premennych (14) a 2000 BDD diagramov trval 19.320000 sekund
Priemerna miera redukovanosti BDD diagramov bola 22.45%
```

```
PS C:\Users\Dzinak\Documents\FIIT\4. Semester\DSA\Zadania\zadanie3> cd "c:\Users
Cely test pre pocet premennych (15) a 2000 BDD diagramov trval 39.101000 sekund
Priemerna miera redukovanosti BDD diagramov bola 22.37%
```