Richard Šléher
ID: 120890

Dátové štruktúry a algoritmy

Zadanie č. 2

Richard Šléher
ID: 120890

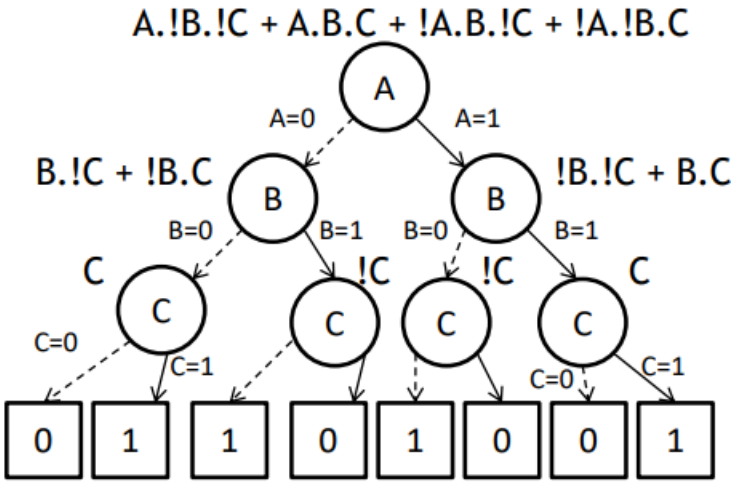
Obsah

Richard Šléher
ID: 120890

1. Binárne rozhodovacie diagramy

1.1 Fungovanie binárneho rozhodovacieho diagramu

Binárny rozhodovací diagram (BDD) je dátovou štruktúrou, ktorá obsahuje root v ktorom je uložená vstupná formula. Na každý uzol napájajú 2 child nody. Low child pre hodnotu 0 a high child pre hodnotu 1. V tomto zadaní budeme reprezentovať Booleovské funkcie. Podľa jednotlivých premenných sa strom rozkladá až kým sa nevytvorí list s jeho poslednou premennou alebo negovaným tvarom.



Obrázok: Vizualizácia rozloženia BDD

1.2 Vytvorenie BDD

BDD_create:

Vytvorím 2 nody, ktoré obsahujú iba 1 alebo 0. A vytvorím root node, ktorý pošlem do metódy decompose(), kde sa celá funkcia rozloží rekurzívnym volaním. Po vytvorení stromu sa zavolá metóda reduce_S, ktorá prejde celý strom a odstráni nody s 2 rovnakými child nodmi.

Richard Šléher
ID: 120890

```
public BDD BDD_create(String bfunc, String order) {

    Node trueNode = new Node( bfunkcia: "1", left: null, right: null);
    Node falseNode = new Node( bfunkcia: "0",  left: null, right: null);
    this.order = order;
    nodeMap.put("1", trueNode);
    nodeMap.put("0", falseNode);
    root = new Node(bfunc,  left: null,  right: null);
    numberOfNodes++;
    String[] vars = order.split( regex: "");
    decompose(root, vars,  i: 0);
    reduce_S(root);
    return this;
}
```

```
//tvorenie novych nodov
if (!node.getBfunkcia().equals("0") && !node.getBfunkcia().equals("1")) {
    if (!leftClause.contains("1") && !leftClause.contains("0")) {
        if (node.getLeft() == null) {
            numberOfNodes++;
            Node leftChild = new Node(String.join( delimiter: "+", leftClause),  left: null,  right
            node.setVar(var);
            node.setLeft(leftChild);
            leftChild.setParent(node);
            decompose(leftChild, vars,  i: i + 1);
        }
    }

    if (!rightClause.contains("1") && !rightClause.contains("0")) {
        if (node.getRight() == null) {
            numberOfNodes++;
            Node rightChild = new Node(String.join( delimiter: "+", rightClause),  left: null,  ri
            node.setVar(var);
            node.setRight(rightChild);
            rightChild.setParent(node);
            decompose(rightChild, vars,  i: i + 1);
        }
    }
```

Časť metódy decompose(), kde sa tvoria nové nody.

Richard Šléher
ID: 120890

Decompose:

Volá sa dokým decompose neprejde všetky premenné pre každú vetvu stromu. Na začiatku sa vytvoria dve polia, ktoré reprezentujú low child a high child (v mojej implementácii left a right). Hlavný cyklus prechádza cez jednotlivé klauzuly a rozdeľuje premenné do leftClause a rightClause podľa výskytu. Ak sa v klauzule premenná nenachádza pridáme ju do oboch polí. Ďalej mažeme rovnaké premenné z klauzúl. A po mazaní kontrolujeme či sa celá klauzula nedá vyhodnotiť ako 1 alebo 0. Ak nie tak vytvoríme nový node a voláme decompose(). Ak je 1 alebo 0 tak ju nastavíme na true alebo false node, ktorý sme vytvorili v BDD_create. Na konci sa node redukuje a pridáva do hash tabuľky

Reduce:

Reduce vykonáva redukciu typu I. To znamená, že ak node s rovnakou funkciou existuje v nodeMap tak sa nastaví na vytvorený a netvorí sa nový, ktorý je rovnaký. Inak sa vytvorený node pridá do nodeMap.

```
public Node reduce(Node node){

    Node childNode = nodeMap.get(node.getBfunkcia());

    if (childNode != null) {

        if (node.getParent().getLeft() == node){
            node.getParent().setLeft(childNode);
            return node.getParent().getLeft();
        } else if (node.getParent().getRight() == node){
            node.getParent().setRight(childNode);
            return node.getParent().getRight();
        }
    }
    return node;
}
```

1.3 Nájdenie najvýhodnejšieho BDD

BDD_create_with_best_order:

Metóda má za úlohu vytvoriť N (v mojom prípade 100) stromov s rôznymi postupnosťami premenných. Ak je novo vytvorený strom menší ako uložený tak sa nahradí novo vytvoreným.

Richard Šléher
ID: 120890

```java
public static BDD BDD_create_with_best_order(String bfunc) {

    double bestNumberOfNodes = 0;
    BDD bdd = null;
    BDD bestBDD = null;
    ArrayList<String> usedOrders = new ArrayList<>();
    long totalTime = 0;

    for (int i = 0; i < 100; i++) {

        String order = getUniqueCharacters(bfunc);
        if (!usedOrders.contains(order)) {
            usedOrders.add(order);

            long startTime = System.nanoTime();
            bdd = new BDD().BDD_create(bfunc, order);
            long endTime = System.nanoTime();

            totalTime = (endTime - startTime) / 1000; //microseconds

        } else {
            i--;
        }

        if (bestNumberOfNodes == 0 || bdd.reducedNumberOfNodes < bestNumberOfNodes) {
            bestNumberOfNodes = bdd.reducedNumberOfNodes;
            bestBDD = bdd;
            bdd.time = totalTime;
        }
    }

    return bestBDD;
}
```

getUniqueCharacters:

Metóda slúži na získanie unikátnych charakterov z formuly. Prechádza všetky znaky a kontroluje či sa nachádzajú v poli ak nie pridá ich. Na záver ich zamieša a vytvorí string.

```java
public static String getUniqueCharacters(String input) {
    ArrayList<Character> chars = new ArrayList<>();

    //ziska unikatne premenne
    for (int i = 0; i < input.length(); i++) {
        char c = input.charAt(i);

        if (c != '+' && c != '!' && !chars.contains(c)) {
            chars.add(c);
        }
    }

    //zamiesa ich a vytvori postupnost
    Collections.shuffle(chars);

    StringBuilder sb = new StringBuilder();
    for (char c : chars) {
        sb.append(c);
    }

    return sb.toString();
}
```

1.4 BDD_use

Slúži na overenie či bol daný strom správne vytvorený. Najprv zistí správny výsledok z formuly a uloží jeho hodnotu. Overuje ho tak, že za premenné dosadí 1 a 0 podľa vstupu. Potom cyklom prejdem jednotlivé klauzuly dokým nenájdem jednu bez 0 alebo dokým ich neprejdem všetky. Môj strom získa char podľa ktorého bola formula rozkladaná a zistí jej hodnotu a podľa toho sa posunie vľavo alebo vpravo v strome. Výsledok sa uloží a potom sa oba porovnajú. Ak sa nerovnajú metóda vráti -1 ak sa rovnajú tak vráti 1 alebo 0 podľa pravdivosti.

Richard Šléher
ID: 120890

```java
public String BDD_use(BDD bdd, String input) {
    //pocitanie spravneho vysledku
    order = bdd.order;

    String flag = "-1";
    String[] vars = order.split( regex: "");
    String clause = root.getBfunkcia();
    Node node = root;

    for (int i = 0; i < input.length(); i++) {
        if (input.charAt(i) == '0') {
            clause = clause.replace( target: "!" + vars[i],  replacement: "1");
            clause = clause.replace(vars[i],  replacement: "0");
        } else if (input.charAt(i) == '1') {
            clause = clause.replace( target: "!" + vars[i],  replacement: "0");
            clause = clause.replace(vars[i],  replacement: "1");
        }
    }

    String[] result = clause.split( regex: "\\+");

    for (String s : result) {
        if (!s.contains("0")) {
            flag = "1";
            break;
        } else {
            flag = "0";
        }
    }
```

```java
    //////////////////////////////////////////////////////////////////////
    //Pocitanie mojho vysledku


    String myFlag = "-2";
    for (int i = 0; i < input.length(); i++) {

        char var = input.charAt(order.indexOf(node.getVar()));

        if (var == '0') {
            if (node.getLeft().getBfunkcia().equals("1")) {
                myFlag = "1";
            } else if (node.getLeft().getBfunkcia().equals("0")) {
                myFlag = "0";
            } else {
                node = node.getLeft();
            }
        } else if (var == '1') {
            if (node.getRight().getBfunkcia().equals("1")) {
                myFlag = "1";
            } else if (node.getRight().getBfunkcia().equals("0")) {
                myFlag = "0";
            } else {
                node = node.getRight();
            }
        }
    }
```

Richard Šléher
ID: 120890

```
    ////////////////////////////////////////////////////
    //Porovnanie vysledkov
    if (flag.equals(myFlag) && flag.equals("1")) {
        return "1";
    }
    else if (flag.equals(myFlag) && flag.equals("0")) {
        return "0";
    } else {
        return "-1";
    }
}
```

2. Testovanie

2.1 Priebeh testovania

Zavolal som najprv metódu generateBfunctions(), ktorá mi vygenerovala 200 funkcií. V tejto metóde sa dá volí koľko rôznych premenných má formula obsahovať, tiež sa dá zvoliť počet premenných v klauzule aj počet klauzúl. Ďalej prvým cyklom prejdem vygenerované formuly a pre každú vytvoríme BDD s náhodným orderom. Každý vytvorený strom prejdeme aj všetkými možnosťami s BDD_use pre kontrolu správnosti stromu. Druhým cyklom vytvoríme best order pre všetky formuly z poľa. Taktiež vytvorené stromy otestujeme s BDD_use.

2.2 Výsledky testovania

Tabuľka pre BDD_create, v ktorej sa nachádzaju priemerné hodnoty z 200 opakovaní pre BDD_create. Čas je uvedený v mikrosekundách a miera zredukovania v percentách. Do počtu nodov som nerátal node s hodnotou 1 alebo 0.

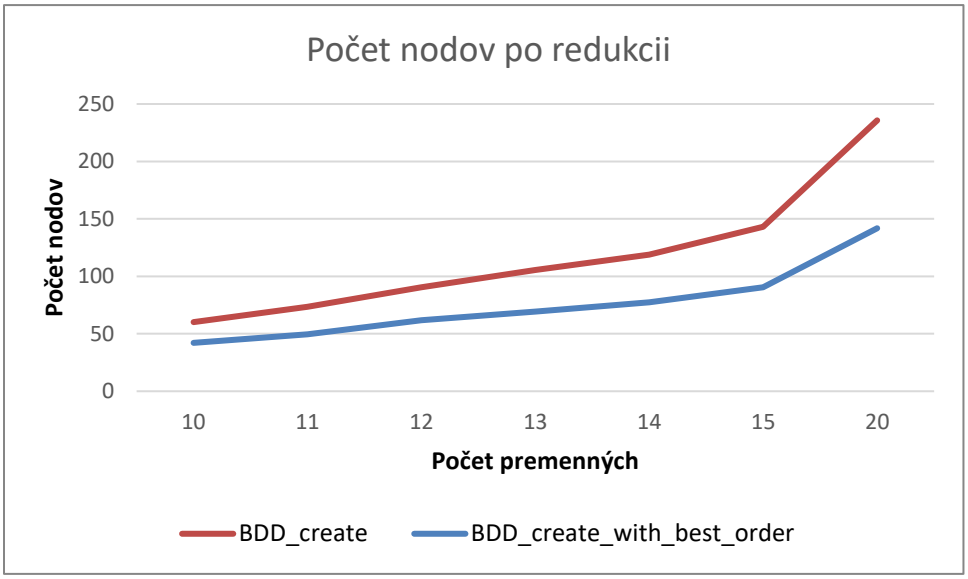
Počet premenných	Čas	Počet nodov	Počet po zredukovaní	Miera zredukovania
10	529,29	323,63	60,17	80,68
11	754,74	549,24	73,55	85,84
12	944,19	1014,39	90,44	90,34
13	1403,26	1723,23	105,57	93,42
14	1787,57	3061,21	118,96	95,67
15	2996,85	5777,33	143,03	97,20
20	34175,35	98483,15	235,73	99,61

Tabuľka pre BDD_create_with_best_order, v ktorej sa ako v predchádzajúcej tabuľke nachádzajú priemerné hodnoty z 200 opakovaní pri tvorení best orderu. Čas vytvorenia je pre najlepší nájdený strom.

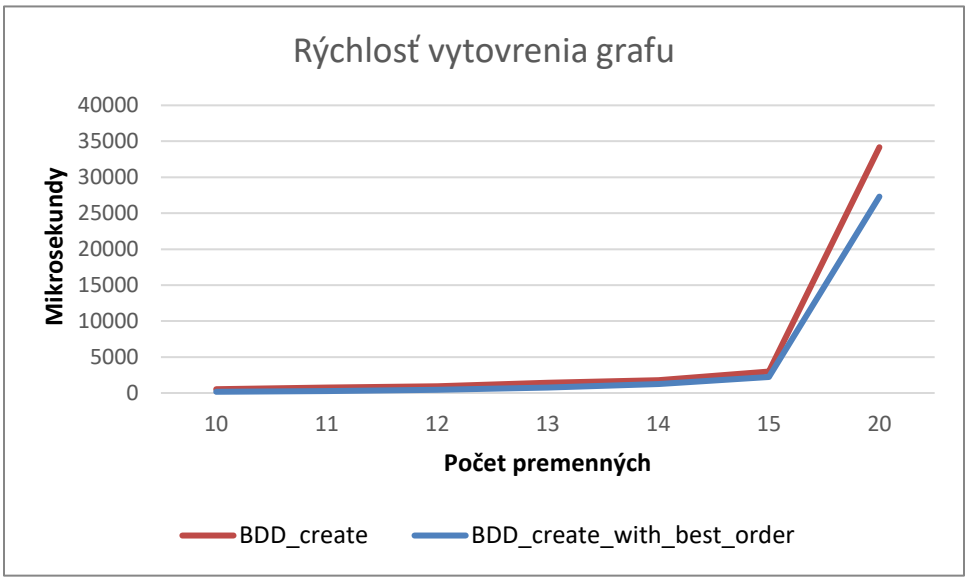
Počet premenných	Čas	Počet nodov	Počet po zredukovaní	Miera zredukovania
10	172,14	297,17	42,12	85,01
11	262,25	500,96	49,59	89,24
12	439,39	927,85	61,87	92,44
13	739,13	1569,34	69,34	94,90
14	1249,33	2738,41	77,49	96,64
15	2212,23	5358,54	90,53	98,01
20	27306,18	83533,20	141,85	99,71

Richard Šléher
ID: 120890

Graf nám znázorňuje priemerný počet nodov pre vytvorené stromy. Červena krivka je pre samotný BDD_create. Vidíme, že BDD_with_best_order nám naozaj vrátil výhodnejší strom.

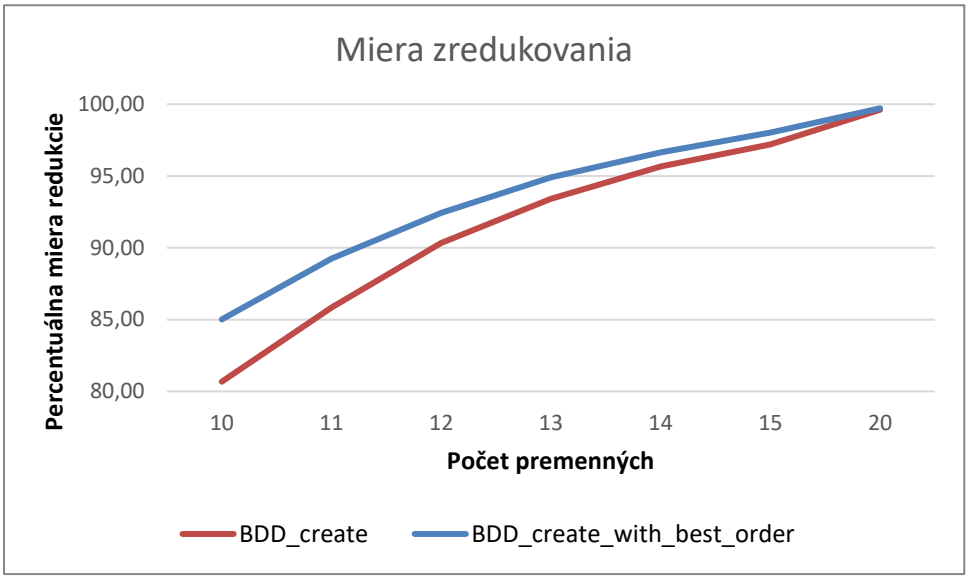


Z grafu môžeme vidieť rýchlosť tvorenia stromov. Opäť je BDD_with_best_order rýchlejší pre nájdený strom. Celkovo je však hľadanie najlepšieho stromu pomalšie, lebo sa generuje 100 stromov ktoré sa porovnávajú. Pri BDD_create sa tvorí vždy iba jeden čiže spolu 200 a pri best order spolu 20 000 stromov.



Richard Šléher
ID: 120890

BDD_create_with_best_order má vyššiu mieru zredukovania ako BDD_create.



3. Výsledok

Z grafu vidíme, že priemerná rýchlosť tvorenia BDD je O(2^n). Je to kvôli počtu premenných, lebo každou premennou sa pridáva ďalšia vrstva stromu, v ktorej sa nachádza dvojnásobok predošlej vrstvy. Priestorová zložitosť je pre najhorší prípad O(2^n) ale s redukciou to je O(log n). Z môjho grafu to nie je dobre vidieť, lebo som tam spravil skok medzi 15 a 20 premennými.