DSA – Zadanie 2

Andrej Stuchlý

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

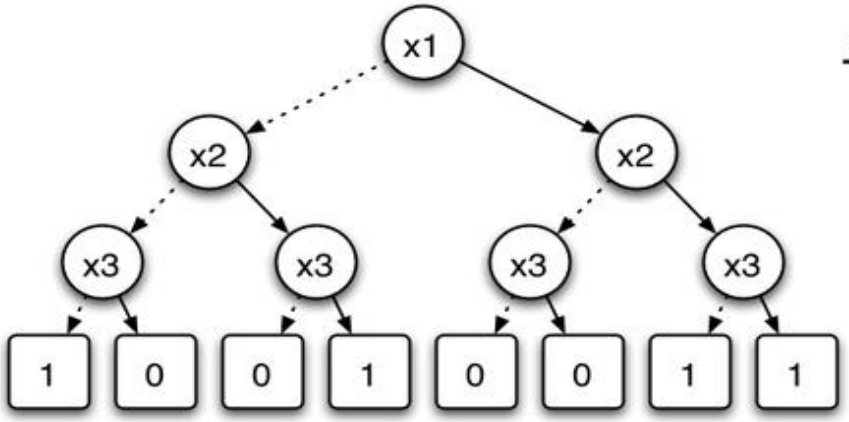
2023

Zadanie

Našou úlohou v tomto zadaní bolo porovnať vytvoriť binárny vyhľadávací diagram pre ľubovoľnú booleovskú funkciu. Konkrétne bolo potrebné implementovať nasledujúce 3 funkcie: BDD_create, BDD_create_with_best_order, BDD_use. Taktiež bolo treba otestovať korektnosť programu, či vracia pre jednotlivé vstupy do booleovských funkcií správny výstup, a s akou časovou a pamäťovou efektívnosťou program pracuje.

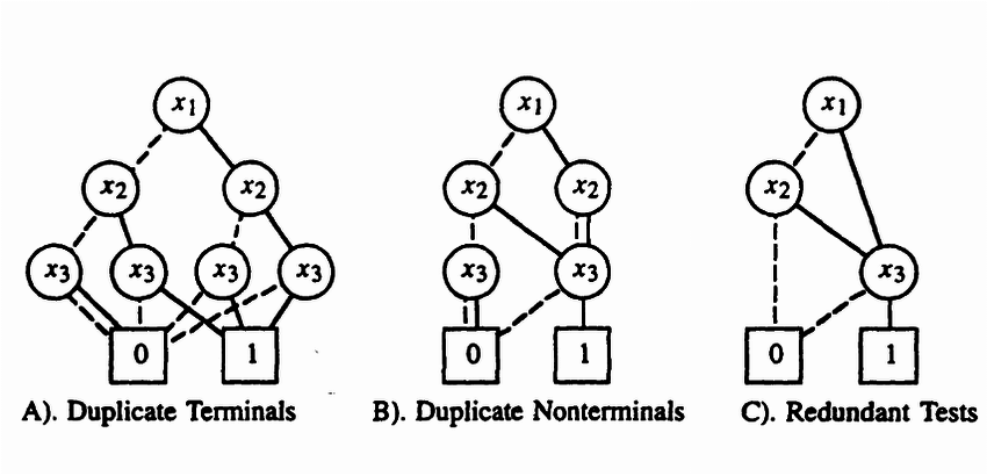
BDD štruktúra

BDD (Binary Decision Diagram) je dátová štruktúra používaná v informatike na reprezentáciu booleovských funkcií. Jej hlavným cieľom je poskytnúť efektívny spôsob reprezentácie a manipulácie s týmito funkciami. BDD predstavuje binárny strom, ktorý reprezentuje rozhodovacie pravidlá vo forme funkcií AND a OR. Vyhodnocovanie týchto funkcií je založené na jednoduchom pravidle, ktoré určuje, akým smerom sa má strom prechádzať.



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

Táto štruktúra sa taktiež vyznačuje redukcia nodov v BDD (Binary Decision Diagram) je proces, ktorý sa používa na zjednodušenie a optimalizáciu diagramov BDD. Cieľom redukcie nodov je znížiť počet uzlov v BDD a tým zlepšiť jeho výkon a efektivitu. Redukcia nodov sa dosahuje spojením niektorých uzlov, ktoré majú rovnaké formuly.



Implementácia

Hlavnou triedou je BDD, ktorá obsahuje informácie o koreni BDD, poradí premenných, počte premenných a veľkosti BDD. V triede sa definujú aj dva základné nody (uzly) - nulový a jednotkový, samotný strom sa skladá z BDDNodov.

```
class BDDNode:
    def __init__(self, variable, formula):
        self.variable = variable  # A B C None
        self.formula = formula  # A+B.C+D
        self.left = None  # 0
        self.right = None  # 1


class BDD:
    zero_node = BDDNode(None, "0")
    one_node = BDDNode(None, "1")

    def __init__(self, order, variables_amount):
        self.root = None
        self.order = order  # [C, B, A, D]
        self.size = 2
        self.variables_amount = variables_amount
```

Hlavnou funkciou je BDD_create_with_best_order(b_function), ktorá vytvorí BDD pre booleanovskú funkciu zadanú ako reťazec b_function. Funkcia postupne skúša rôzne poradia premenných a vyberá to s najmenšou veľkosťou BDD.

```
def BDD_create_with_best_order(b_function):
    current_best = None
    current_least = sys.maxsize
    order = sorted(set(c for c in b_function if c.isalpha()))
    for i in range(len(order)):
        created_bdd = BDD_create(b_function, order[:])
        if created_bdd.size < current_least:
            current_best = created_bdd
            current_least = current_best.size
        order = rotate_right(order)
        #random.shuffle(order)
    return current_best
```

Vytvorenie BDD pre konkrétne poradie premenných je implementované funkciou BDD_create(). Táto funkcia rekurentne rozkladá funkciu na menšie formuly a vytvára BDD uzly pre každý z nich. Keď sa nájde rovnaký formulu, použije sa už vytvorený uzol, aby sa zabránilo duplikácii uzlov.

```
def BDD_create_rec(bdd, formula, order, index, used_nodes):
    if formula == "1":
        return bdd.one_node
    if formula == "0":
        return bdd.zero_node
    if formula in used_nodes:
        return used_nodes[formula]

    var = order[index]
    pos_formula = formula.replace(var, "1")
    if pos_formula == formula:
        return BDD_create_rec(bdd, formula, order, index + 1, used_nodes)
    pos_formula = simplify_boolean(pos_formula)
    neg_formula = formula.replace(var, "0")
    neg_formula = simplify_boolean(neg_formula)

    bdd.size += 1
    node = BDDNode(var, formula)
    used_nodes[formula] = node
    node.right = BDD_create_rec(bdd, pos_formula, order, index + 1, used_nodes)
    node.left = BDD_create_rec(bdd, neg_formula, order, index + 1, used_nodes)

    return node
```

Pre skrátenie veľkosti BDD a zvýšenie efektivity sa používa funkcia simplify_boolean(), ktorá redukuje booleanovské výrazy na ich najjednoduchšie formy. V ďalších funkciách sa používa prechádzanie BDD na základe hodnôt premenných a následné hľadanie odpovedajúceho booleanovského výrazu.

```
def simplify_boolean(expr):
    old_expr = expr
    expr = replace_negation(expr)

    patterns = {
        r'[A-Z]\.0': '0',
        r'0\.[A-Z]': '0',
        r'[A-Z]\.1': lambda match: match.group(0)[0],
        r'1\.[A-Z]': lambda match: match.group(0)[2],
        r'[A-Z]\+0': lambda match: match.group(0)[0],
        r'0\+[A-Z]': lambda match: match.group(0)[2],
        r'[A-Z]\+1': '1',
        r'1\+[A-Z]': '1',
        r'0\+0': '0',
        r'0\.0': '0',
        r'1\+1': '1',
        r'1\.1': '1',
        r'0\+1': '1',
        r'1\+0': '1',
        r'0\.1': '0',
        r'1\.0': '0',
    }

    for pattern, replacement in patterns.items():
        expr = re.sub(pattern, replacement, expr)

    if old_expr != expr:
        expr = simplify_boolean(expr)
    return expr
```

Na overenie správnosti a aj využiteľnosti tejto implementácie štruktúry sa použije funkcia BDD_use, ktorá zoberie diagram a poradie hodnôt (0 alebo 1) a vráti výslednú pravdivostnú hodnotu po dosadení poradia hodnôt do diagramu.

```python
def BDD_use(bdd, inputs):
    current = bdd.root
    while current.variable is not None and (current.formula != "0" and current.formula != "1"):
        var_index = bdd.order.index(current.variable)
        if inputs[var_index] == 0:
            current = current.left
        else:
            current = current.right
    return current.formula
```

Zložitosti

Časová zložitosť

BDD_create - 2^n (n je počet premenných)

BDD_create_with_best_order - 2^n*n (n je počet premenných)

BDD_use – n (n je počet premenných)

Pamäťová zložitosť

BDD_create - 2^n (n je počet premenných)

BDD_create_with_best_order - 2^n*n (n je počet premenných)

BDD_use – n (n je počet premenných)

Testovanie

Pri testovaní som sa zameral najmä na časy a korektnosť riešenia. Vytvoril som test pre základnú funkcionalitu, ktorá ukazuje optimálne aj neoptimálne poradie na vytváranie a dosadenie hodnôt do ideálneho poradia. Druhá časť testov sa skladá zo 100 vygenerovaných formúl, pre ktoré N-krát (N – počet premenných) vytvorím poradie, vyberiem to najlepšie a použijem v ňom vygenerované hodnoty jednotiek a núl. Zároveň si ukladám celkovú redukciu nodov v percentách a na konci ju vypíšem spolu s celkovým časom trvania.

```python
def test2():
    print("\n\nMain test:\n")
    variables = 13
    max_nodes = 2**variables - 1 + 2
    formulas = 100
    start_time = time.time()
    average_reduction = 0
    for i in range(formulas):
        print("Test number " + str(i + 1))
        current_reduction = 0
        generated_formula = generate_formula(variables)
        best_bdd_from_formula = BDD_create_with_best_order(generated_formula)
        values = create_use_values(variables)
        boolean_value = BDD_use(best_bdd_from_formula, values)
        average_reduction += (max_nodes - best_bdd_from_formula.size) / max_nodes * 100
        current_reduction += (max_nodes - best_bdd_from_formula.size) / max_nodes * 100
        print("Formula: " + generated_formula + "\nAmount of variables for the formula: " +
              str(best_bdd_from_formula.variables_amount) + "\nAmount of nodes in the BDD: " +
              str(best_bdd_from_formula.size) + ", the reduction rate is " + str(current_reduction) +
              " %\nFor values " + str(values) + ",\nassigned to order " +
              str(best_bdd_from_formula.order) + ",\nthe return value is " + boolean_value + "\n")
    average_reduction /= formulas
    print("Average reduction = " + str(average_reduction) + " %")
    print("--- Total time:  %s seconds ---" % (time.time() - start_time))
```

```
Test for basic functionality:

Original formula: A.B+C.D+E.F+G.H
Not optimal bdd: 32 nodes with order ['A', 'C', 'E', 'G', 'B', 'D', 'F', 'H']
Least nodes: 10 nodes with order: ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
Boolean value for values [1, 0, 0, 0, 0, 0, 1, 1] : 1


Main test:

Test number 1
Formula: L.G.I+!C.B.H.!K.!B+!A+!B+B+!K+!A+!K.!D.K.K.!C.!F+!A+!E+!K+!C.B.!F.!F.C+!M+M+I+C.J
Amount of variables for the formula: 13
Amount of nodes in the BDD: 49, the reduction rate is 99.4019284755279 %
For values [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1],
assigned to order ['K', 'L', 'M', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'],
the return value is 1

Test number 2
Formula: B+!K.!D+M+H.M.!M.C+M+!L+!D+G.!E.M+L.!B+G+H.!G.E.H+F.J.!H+A+!F+F+B+I+!A.G+!J.E.!F.M.C.G
Amount of variables for the formula: 13
Amount of nodes in the BDD: 146, the reduction rate is 98.21799096789942 %
For values [1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1],
assigned to order ['F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'A', 'B', 'C', 'D', 'E'],
the return value is 0
```

Výsledky testovania

V tabuľke nižšie je možné vidieť hodnoty testovania s 13, 14 a 15 premennými, pri všetkých som si vygeneroval 100 formúl a rátal celkovú redukciu a potrebný čas na vytvorenie najelpšieho poradia a dosadenia hodnôt do tohto poradia.

Počet premenných	Počet formúl	Priemerná redukcia	Maximálny počet nodov	Priemerný počet nodov	Celkový čas merania
13	100	97.94%	8 193	168	31.78s
14	100	98,34%	16 385	272	57.29s
15	100	98.89%	32 769	363	81.59s
16	100	99.20%	65 537	524	132.16s

Ako je možné vidieť, efektivita zostáva stále rovnaká, dokonca sa so zvyšujúcim počtom premenných po malých hodnotách aj zvyšuje – toto je spôsobené tým, že v mnohých prípadoch je možné vykonať redukciu skoro, čím sa ušetrí veľa nodov. Čo sa týka potrebného času, je vidieť, že pri zvýšení počtu premenných z 15 na 16 došlo k dramatickému zvýšeniu a od 16 premenných vyššie začínajú byť tieto testy dlhé.

Je však potrebné podotknúť, že v prípade vytvárania len jednej formuly by sme čas vydelili stovkou a dostali by sme sa na prijateľnú hodnotu, preto by táto štruktúra pre malé počty formúl bola efektívna aj pri veľkom počte premenných.

Taktiež čo sa týka redukcie uzlov, vždy je to závislé od vygenerovanej formuly – jej dĺžky, operácii v nej a taktiež aj poradia, podľa ktorého sa diagram vytvára, preto môžu vzniknúť mierne odchýlky.

Zhodnotenie

Výsledky testovania ukázali, že program vracia pre každý vstup do booleovskej funkcie korektný výsledok je plne funkčný. Redukcia funguje správne, vyradí mnoho nepotrebných uzlov z BDD, čím sa ušetrí množstvo času, ako pri vytváraní, tak aj prechádzaní diagramu a hľadaní výslednej hodnoty. Diagram dokáže byť veľmi efektívny aj pri väčšom počte nodov, aj keď samozrejme pri stále väčšom počte premenných sa potrebný čas značne zvyšuje.