

Binárny Rozhodovací Diagram

Zadanie 2, Dátové Štruktúry a Algoritmy

Id: 116298, 2021/2022

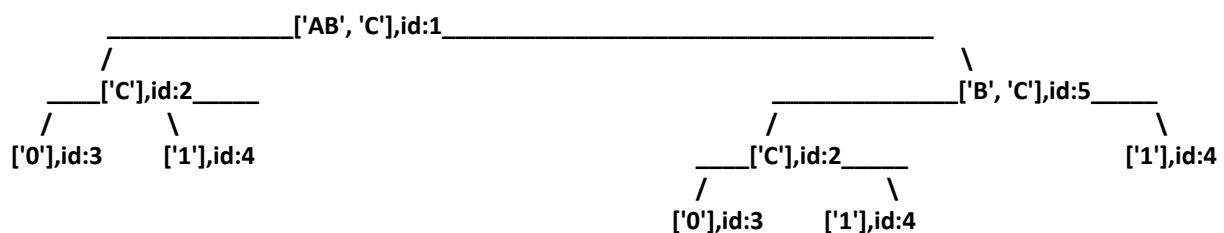
Maximilián Strečanský

Riešenie redukcie:

- Pri vytváraní binárneho diagramu rekurzívne vytvárame najprv ľavé a potom pravé prvky (Inorder). Pred každým pridaním prvku sa najprv pozrieme či daný prvok už v strome neexistuje -> takto docielime redukciu.
- Redukujeme celý strom, nie len samostatný riadok. Po redukcii nám neostanú v strome duplikáty.
- Taktiež ak sa nám obaja potomkovia jedného uzla rovnajú 0 alebo 1, tak zmeníme tento uzol na hodnotu potomka -> takto vymažeme nepotrebné prvky a docielime efektívnejšie prehľadávanie

Riešenie kombinácie:

- Pri funkcii BDD_create prechádzame strom podľa vstupnej kombinácie. Podľa poradia prechádzame strom.
- Príklad booleovskej funkcie $AB+C \rightarrow$ kombinácia 010



- Ako prvé prejdeme do ľavej hodnoty C keďže keď za A dosadíme 0 dostaneme C. Ďalej keď dosadíme za B 1 ostaneme stále v rovnakom uzle a až podľa hodnoty C ovplyvníme výsledok
- Takto dokážeme korektne redukovať strom na najmenšie množstvo prvkov

Vlastné Štruktúry:

```
# Trieda jednej nody
class Node(object):
    def __init__(self, value, i):
        self.left = None
        self.right = None
        self.value = value
        self.id = i
```

- V triede NODE ukladáme pravý a ľavý prvok, hodnotu prvku a unikátne id pomocou ktorého vieme pri vypisovaní prvkov vidieť, že prvky sú rovnaké

```
# Trida binarneho rozhodovacieho diagramu
class BDD(object):
    def __init__(self, poradie, fList):
        self.root = Node(fList, 1)
        self.values = 1
        self.poradie = poradie
        self.pocetPremennych = len(poradie)
```

- V triede BDD si ukladáme smerník na počiatočnú hodnotu binárneho diagramu ako aj počet unikátnych uzlov diagramu, poradie v tvare (ABC...) a počet premenných.

Jednotlivé funkcie

```
# Metoda pre vypis vysledku
def BDD_use(self, combination):
    tempRoot = self.root
    for i in range(0, len(combination)):
        number = combination[i]
        if "1" in tempRoot.value:
            return "1"
        if "0" in tempRoot.value:
            return "0"
        if self.poradie[i] in listToString(tempRoot.value):
            if number == "0":
                tempRoot = tempRoot.left
            elif number == "1":
                tempRoot = tempRoot.right
            else:
                return "-1"
        else:
            continue
    if "1" in tempRoot.value:
        return "1"
    return "0"
```

- Na vstupe dostaneme kombináciu v tvare 000, 010, 110... a binárny diagram. Cyklicky prejdeme cez uzle diagramu v závislosti od kombinácie. Ak dostaneme na vstupe 0 posunieme sa doľava, ak 1 doprava. Najprv sa ale pozrieme, či sa v NODE nachádza písmeno poradia. Ak áno, posunieme sa podľa kombinácie a ak nie, ostaneme v NODE.

```

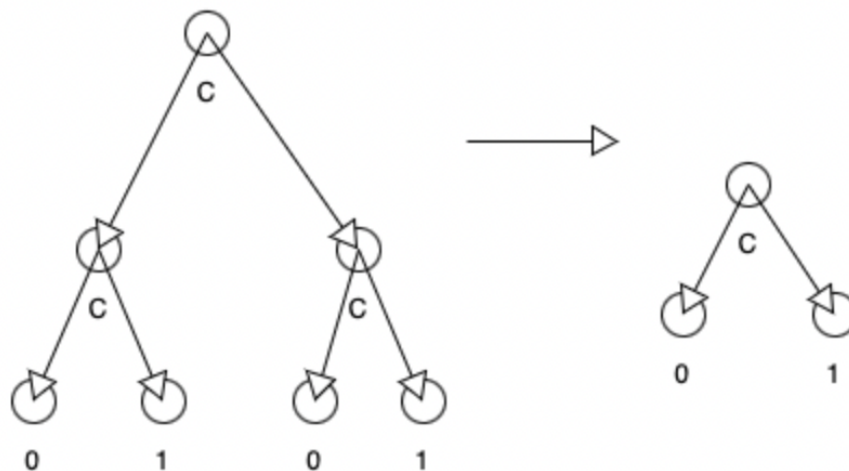
# Funkcia na vytvorenie binarneho diagramu
def BDD_create(self, root, poradie, bfunkcia):
    if poradie:
        tempLeft = leftString(bfunkcia, poradie[0])
        tempRight = rightString(bfunkcia, poradie[0])
        # Pripad kedy je root rovnaky s pravou a lavou stranou
        # Prejdeme na dalsiu instanciu
        if root.value == tempLeft and root.value == tempRight:
            root = self.BDD_create(root, poradie[1:], root.
value)
        # Ak mame pravu a lavu stranu rovnaku
        elif tempLeft == tempRight:
            answerRoot = checkForDuplicates(self.root,
tempLeft, None)

        # Ak sa hodnota uz nachadza v strome iba ju nastavime ako lavy
        prvok
            if answerRoot:
                root.left = answerRoot
            else:

        # Ak sa v nom nachadza 1 alebo 0 -> nepokracujeme v nasledovno
        m vytvarani
            if "1" in tempLeft or "0" in tempLeft:
                root.left = Node(tempLeft, self.incValues(
1))
            else:
                root.left = Node(tempLeft, self.incValues(
1))
                root.left = self.BDD_create(root.left,
poradie[1:], tempLeft)
                root.right = root.left

```

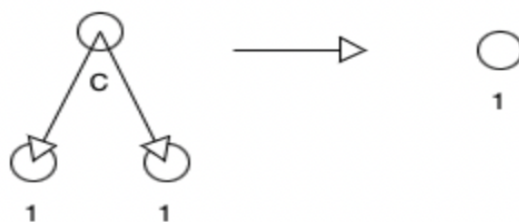
- Vo funkcii BDD_create máme na vstupe BDD diagram, aktuálnu NODE, poradie a booleovskú funkciu. Ako prvé sa pozrieme či existuje v poradí nejaký prvok, ak nie, vieme, že sme na konci diagramu ak áno pokračujeme. Rozložíme si booleovskú funkciu na pravý a ľavý prvok a postupne prechádzame. Ako prvé máme prípad kedy máme aktuálny NODE rovnaký s pravým a ľavým. Vrátime ďalšiu inštanciu



- Ďalší prípad je keď máme pravú a ľavú stranu rovnakú, takto sa pozrieme či sa v strome nenachádza ďalšia rovnaká hodnota. Ak nie, nastavíme pravý prvok na ľavý a pokračujeme v ľavom prvku

```
# Obyčajny prípad, najprv sa pozrieme na ľavý prvok
answerRoot = checkForDuplicates(self.root, tempLeft, None)
# Ak sa hodnota už nachádza v strome iba ju nastavíme ako ľavý prvok
if answerRoot:
    • root.left = answerRoot
else:
    # Ak sa v nom nachádza 1 alebo 0 -> nepokračujeme v nasledovnom vytvaraní
    if "1" in tempLeft or "0" in tempLeft:
        root.left = Node(tempLeft, self.incValues(1))
    else:
        root.left = Node(tempLeft, self.incValues(1))
        root.left = self.BDD_create(root.left, poradie[1:], tempLeft)
```

- Tretí prípad je obyčajný, najprv sa snažíme nájsť duplikát a ak ho nenájde, spustíme BDD_create na ľavý prvok. Ak sa v aktuálnej NODE nachádza 1 alebo 0 toto robiť nemusíme
- Rovnako toto spravíme pre pravý prvok
- Ako posledný krok sa pozrieme či nie je pravý a ľavý potomok rovnaký s hodnotami 1 alebo 0 ak áno, nastavíme aktuálny uzol na hodnotu ľavého



- Keďže sa vždy pred pridaním prvku pozrieme, či sa už daná hodnota nenachádza v strome, docielime takto redukciu. V niektorých prípadoch ani nemusíme pokračovať v pridávaní potomkov.

Spôsob testovania