Binárny Rozhodovací Diagram

Zadanie 2, Dátové Štruktúry a Algoritmy

Id: 116298, 2021/2022

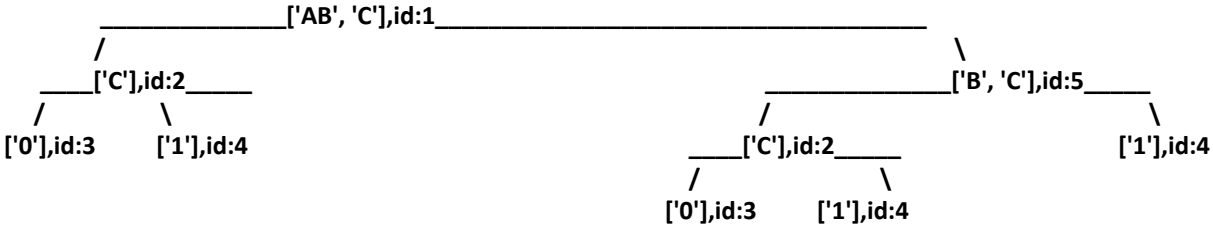
Maximilián Strečanský

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 redukcií 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 -> 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:

```python
# 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

```python
# 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.

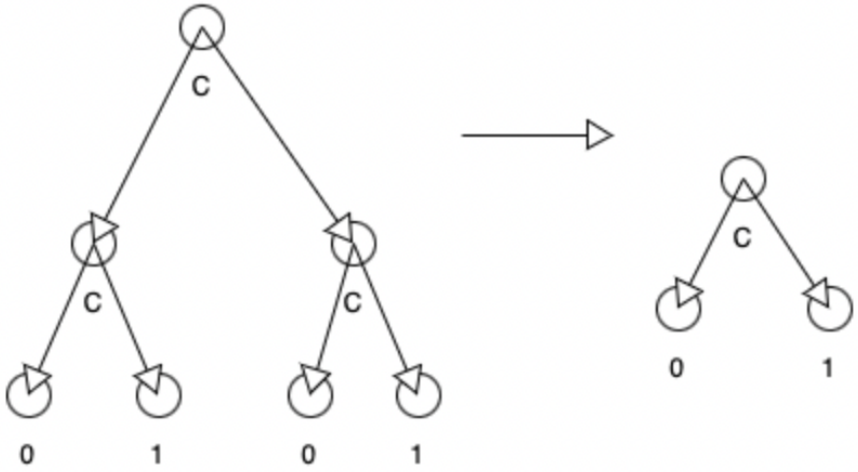
Jednotlivé funkcie

```
# 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
```

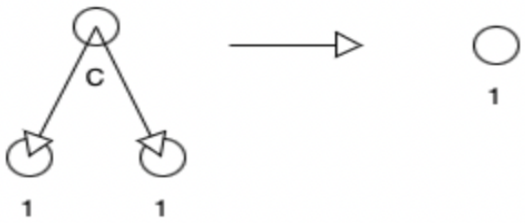
- Vo funkcii BDD_create máme na vstupe BDD diagram, aktuálnu NODE, poradie a booleovskú funkciu. Ako prvé sa pozrieme či existuje v poradí nejaký prvok, ak nie, vieme, že sme na konci diagramu ak áno pokračujeme. Rozložíme si booleovskú funkciu na pravý a ľavý prvok a postupne prechádzame. Ako prvé máme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

```
# Obycajny pripad, najprv sa pozrieme na lavy prvok
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 nasledovnom 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)
```

- Tretí prípad je obyčajný, najprv sa snažíme nájsť duplikát a ak ho nenájdeme, 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.

Náhodné generovanie B-funkcií

```
# Metoda na vytvorenie nahodnej boolovskej funkci
e
def createRandomFunction(velkost, dlzka):
    abeceda = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    temp = ""
    cislo = 0
    i = 0
    while i < dlzka:
        if cislo >= velkost - 1:
            temp += "+"
            cislo = 0
        if random.randint(0, 2) == 0:
            temp += "!"
        cislo = random.randint(cislo, velkost - 1
)
        temp += abeceda[cislo]
        cislo += 1
        i += 1
    for j in range(0, velkost - 1):
        if abeceda[j] not in temp:
            if random.randint(0, 1) == 0:
                temp += "!"
            temp += abeceda[j]
    return temp
```

- Na vstupe dostaneme veľkosť, ktorá určuje, koľko premenných sa použije a dĺžku, ktorá určuje ako dlhá bude B-funkcia
- Cyklicky generujeme náhodné písmená. Aby sme docielili abecedne zoradené náhodné funkcie, máme podmienku, ktorá nám hovorí, že ak je posledné písmeno v jednom prvku väčšie ako dĺžka, pridáme na koniec +.
- Druhá podmienka nám náhodne generuje negáciu

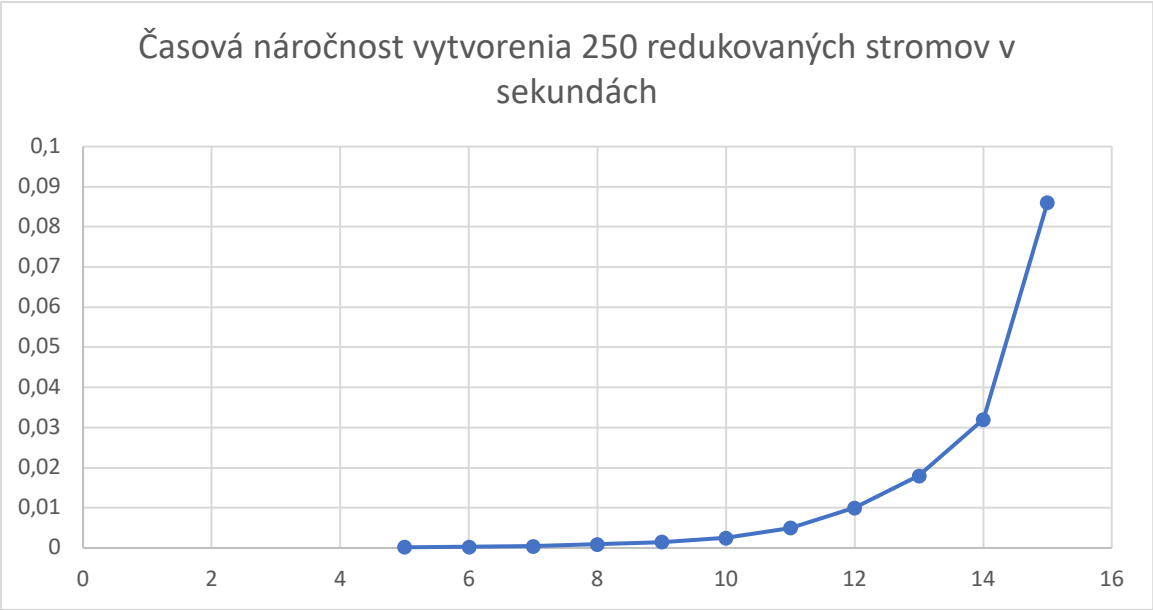
- Na koniec sa pozrieme, ci sa všetky písmená poradia nachádzajú vo funkcií a ak nie, tak ich pridáme

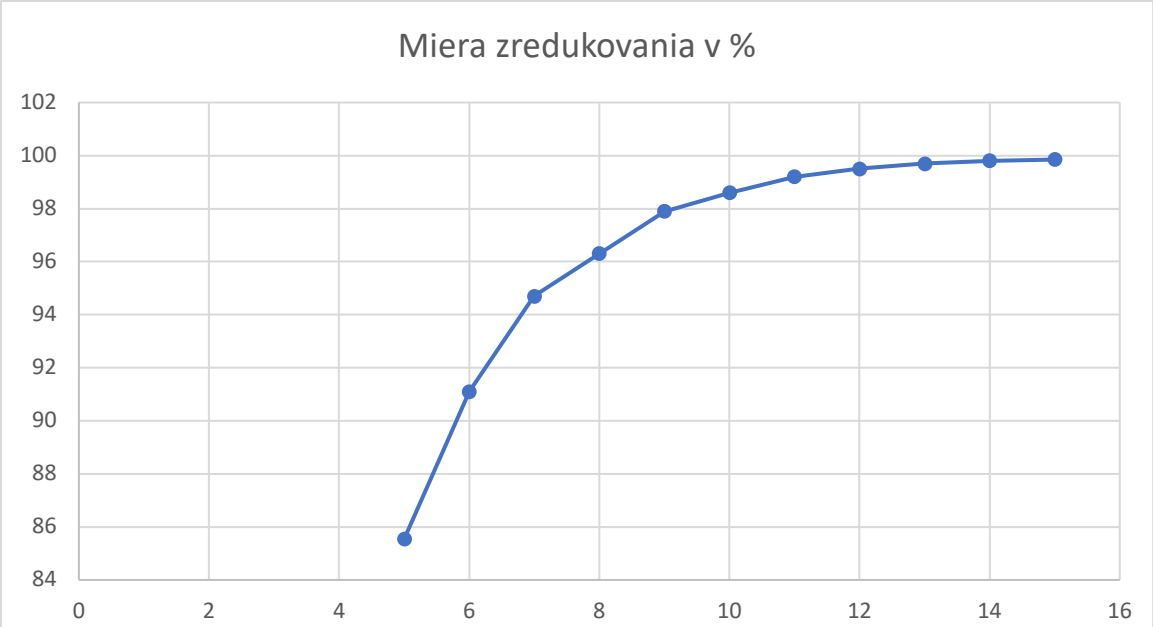
Spôsob testovania

```
for velkost in range(5, 16):
        pocet1 = 0
        pocet2 = 0
        timecounter = 0
        testtimecounter = 0
        maxsize = 0
        for j in range(0, 251):
            dlzka = velkost * 2
            temp = createRandomFunction(velkost, dlzka)
            bfunkcia = temp.split("+")
            poradie = getPoradie(temp)

            tracemalloc.start()
            start_time = time.time()
            good = BDD(poradie, bfunkcia)
            good.root = good.BDD_create(good.root, poradie, bfunkcia)
            maxsize += int(tracemalloc.get_traced_memory()[1])
            end_time = time.time()
            tracemalloc.stop()

            bad = BDD(poradie, bfunkcia)
            bad.root = bad.BDD_createWithDuplicates(bad.root, poradie)

            temp = createCombinations("", velkost, [])
            test_start_time = time.time()
            for item in temp:
                good.BDD_use(item)
            test_end_time = time.time()

            pocet1 += good.values
            pocet2 += bad.values
            timecounter += end_time - start_time
            testtimecounter += test_end_time - test_start_time
```

- 15x vygenerujeme 250 funkcií. Dĺžka bude dvojnásobok veľkosti. Meriame časy pre vytvorenie redukovaného stromu a pre prehľadanie každej možnosti.
- Pri vytváraní redukovaného BDD meriame využitie pamäte
- Na konci vypíšeme mieru zredukovania, využitie pamäte a časy.

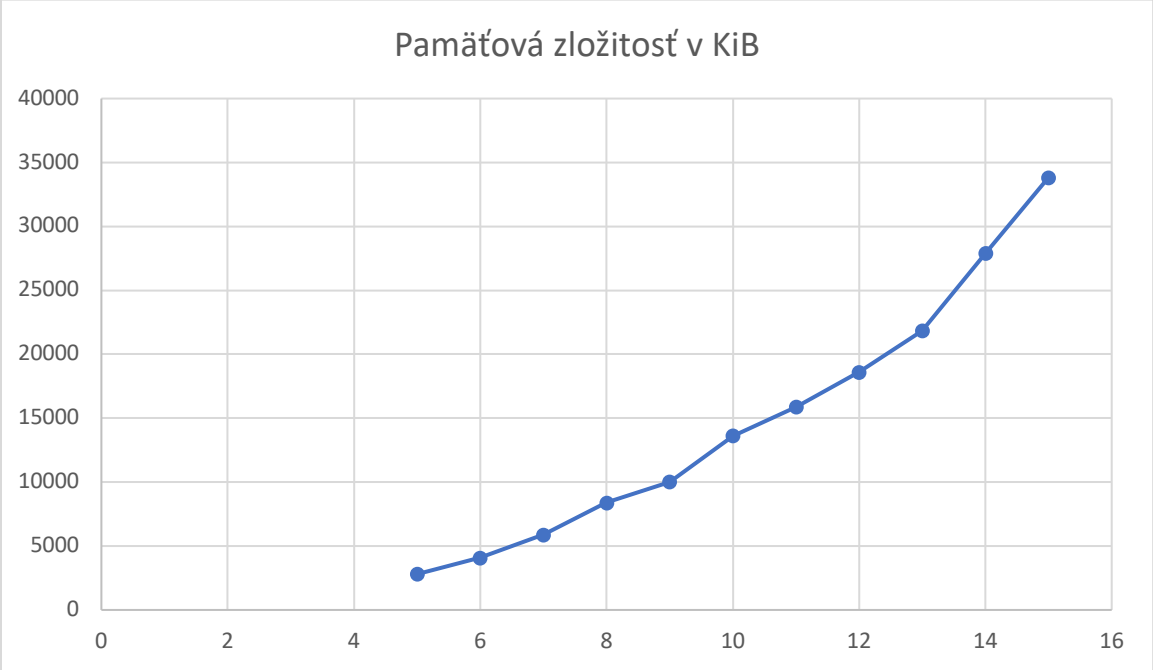
Časová zložitosť





- Odhadovaná zložitosť pre vytvorenie redukovaného stromu je:
  $O(n^2)$, kde n je dĺžka poradia.
- Odhadovaná zložitosť pre prejdenie všetkých výsledkov je:
  $O(2^n)$, kde n je dĺžka poradia.

Pamäťová zložitosť



- Pri mojom riešení sa pri redukcií prehľadáva celý strom namiesto toho, aby sme si dané hodnoty ukladali -> Takýmto postupom dostaneme čo najmenšie zaplnenie pamäte

Záver:

Cieľom tejto práce bolo vytvoriť redukovaný binárny rozhodovací diagram. Spôsob redukcie bol nasledovný: pred pridaním prvku do stromu sa pozrieme na všetky hodnoty v strome a novú hodnotu pridáme ak sa v strome nenachádza, inak nastavíme smerník na nájdený uzol. Druhý prípad je absorpcia -> ak má uzol rovnakých potomkov tak sa uzol rovná potomkovi.

Vďaka tomuto riešeniu dosiahneme najväčšiu možnú redukciu so zameraním na pamäťovú náročnosť. Keďže nemusíme ukladať nadbytočné prvky, docielime v tomto veľkú efektivitu.