STU FIIT

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Dátové štruktúry a algoritmy

Binárne Rozhodovacie Diagramy

07. 04. 2021

letný semester 2020/2021

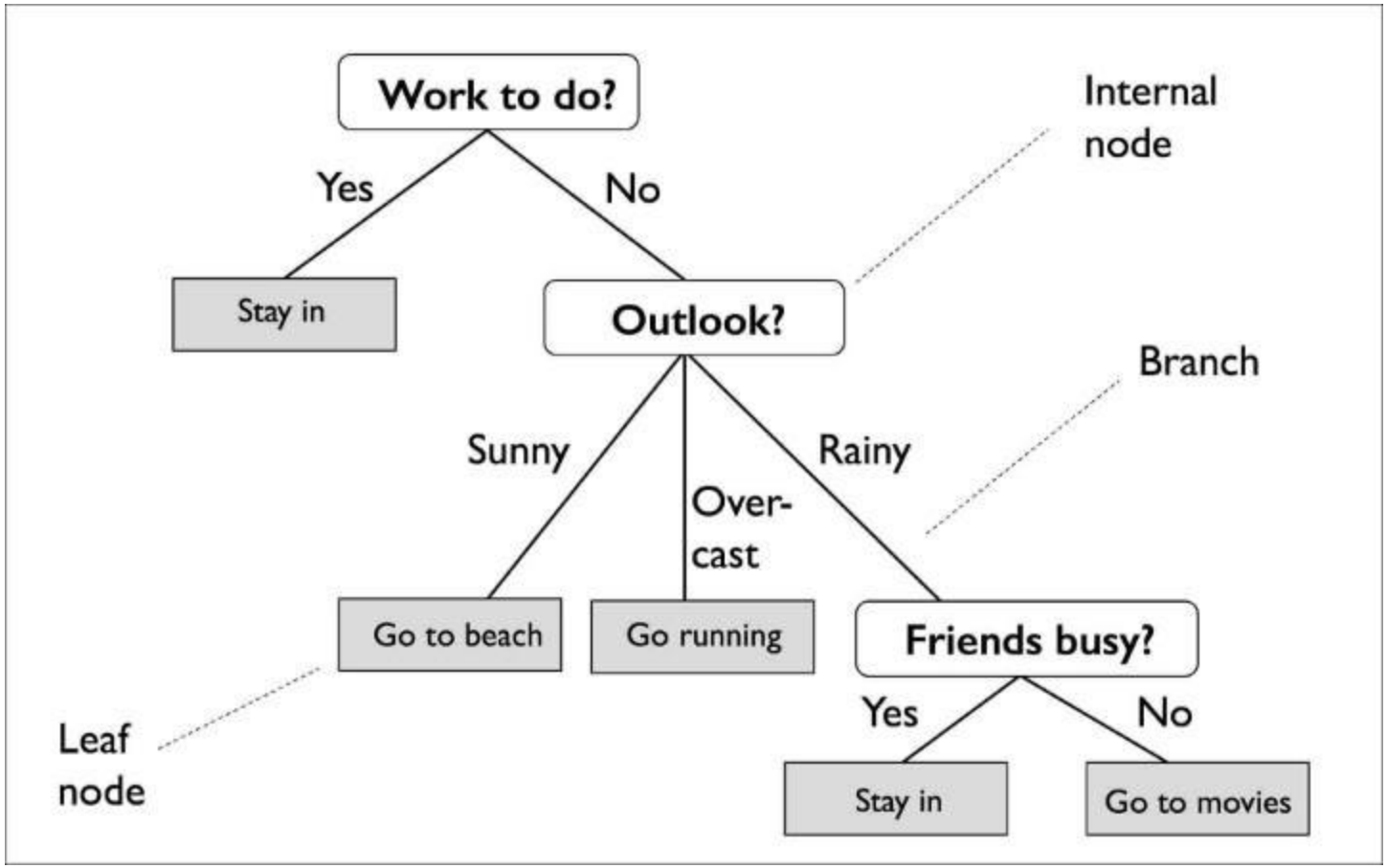
prednášajúci: Lukáš Kohútka

Binárny rozhodovací diagram

- Binárny strom
- Neslúži ako ADT slovník/dynamická množina
- Slúži na rozhodovanie
  - Rozhodovací strom
- Rozhodovanie prebieha prechodom od začiatku stromu (koreň) do konca (list)
- Každý vnútorný uzol predstavuje jedno čiastkové rozhodnutie, list = výsledné rozhodnutie
- Celkové rozhodnutie = celá cesta koreň → list

Rozhodovací strom

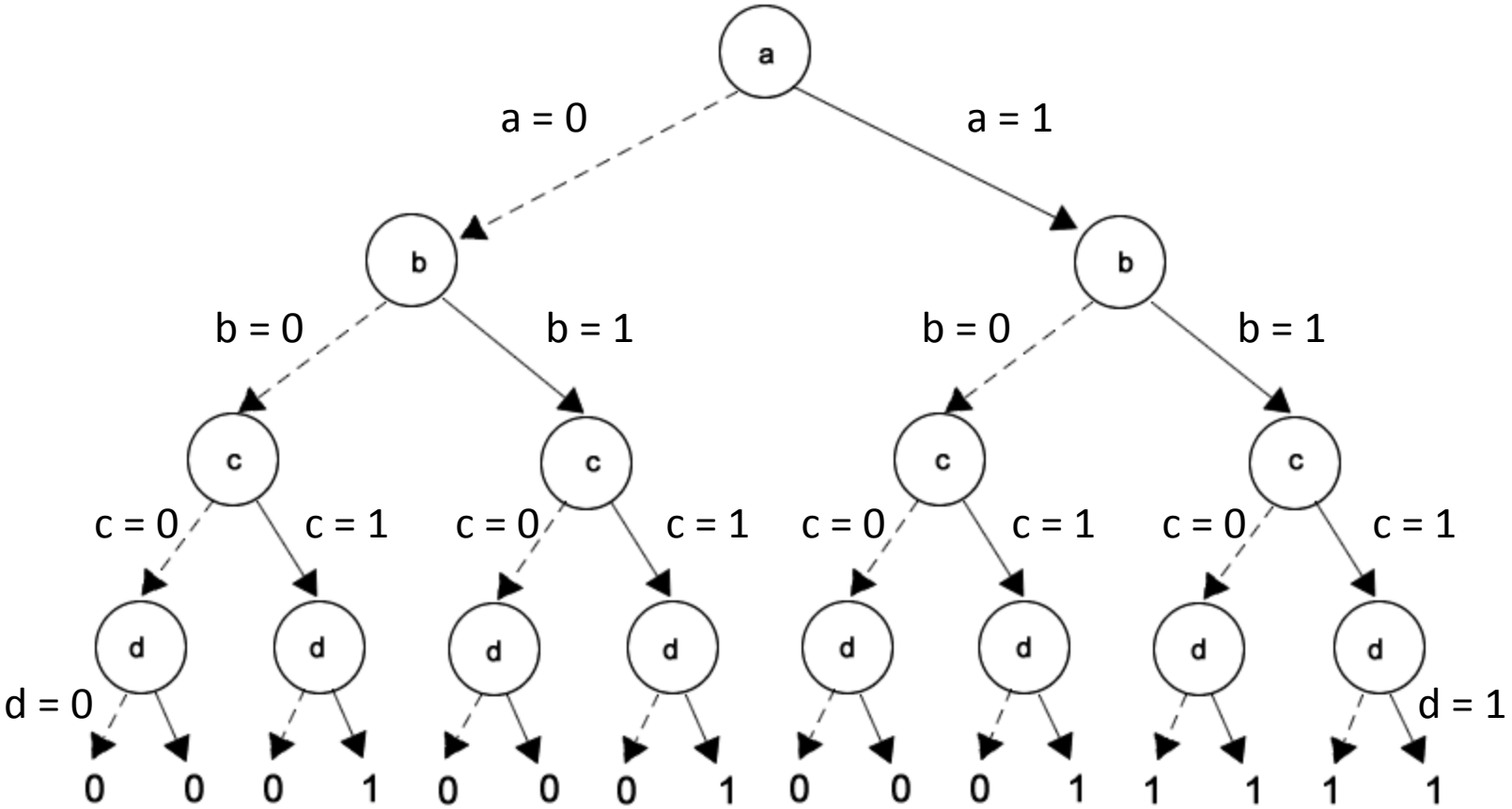
- Môže byť binárny, ale nemusí



Binárny rozhodovací diagram

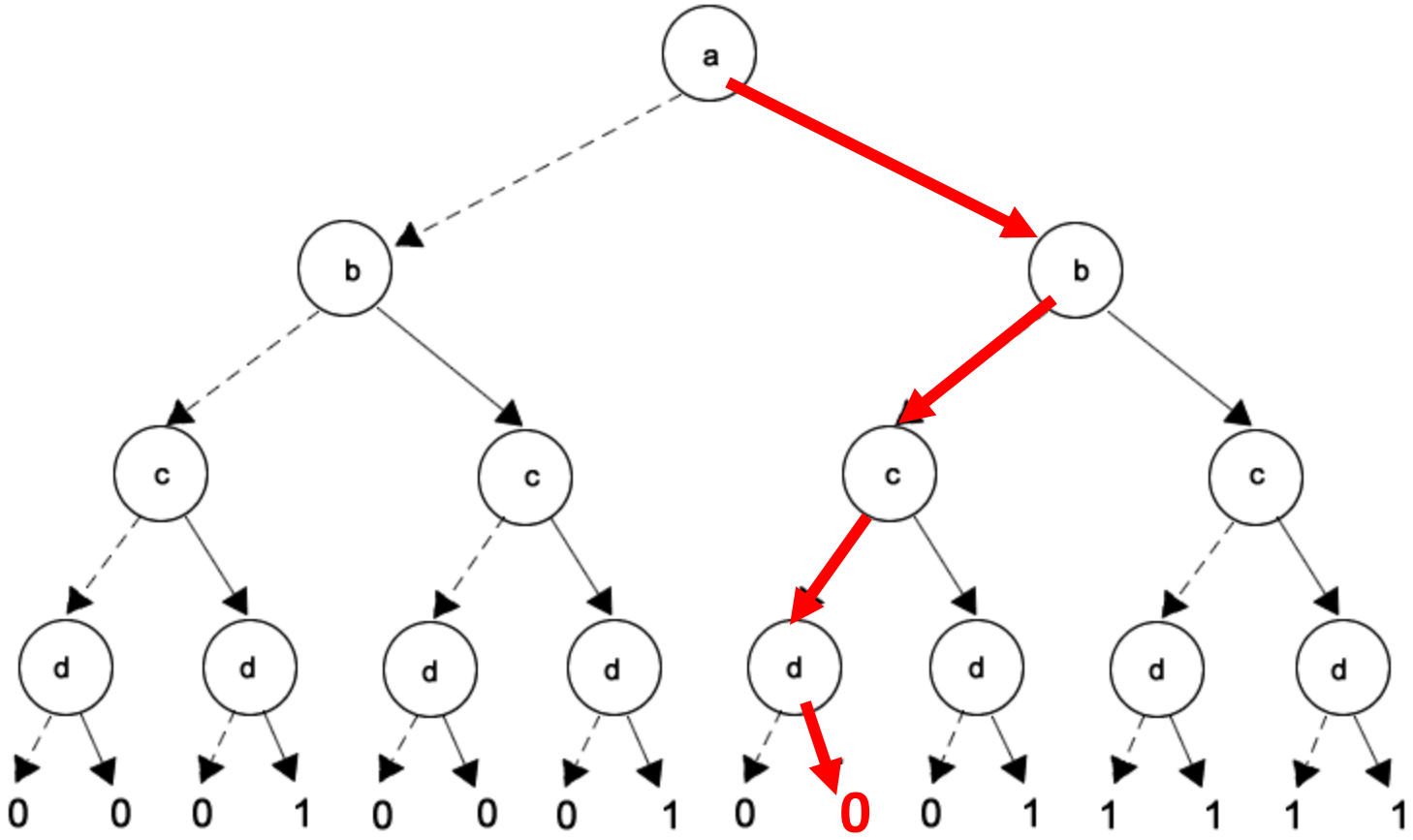
- Binary Decision Diagram (skratka BDD)
- Dátová štruktúra
- Má tvar ako binárny strom
- Slúži na rozhodovanie (rôzne aplikácie)
- Jednou z najpoužívanejších aplikácií je: Reprezentácia ľubovoľnej Booleovskej funkcie

Binárny rozhodovací diagram



Binárny rozhodovací diagram

Aký je výsledok pre a = 1, b = 0, c = 0, d = 1 ?



Reprezentácie Booleovských funkcií

- Pravdivostná tabuľka
- Vektor
  - to isté ako pravdivostná tabuľka, len jej výstupy
- Karnaughova mapa
- Výraz
  - t.j. rovnica (napr. Y = A . B + C)
  - Normálové formy
    - DNF – súčet súčinov
    - KNF – súčin súčtov
- BDD

XOR Truth Table

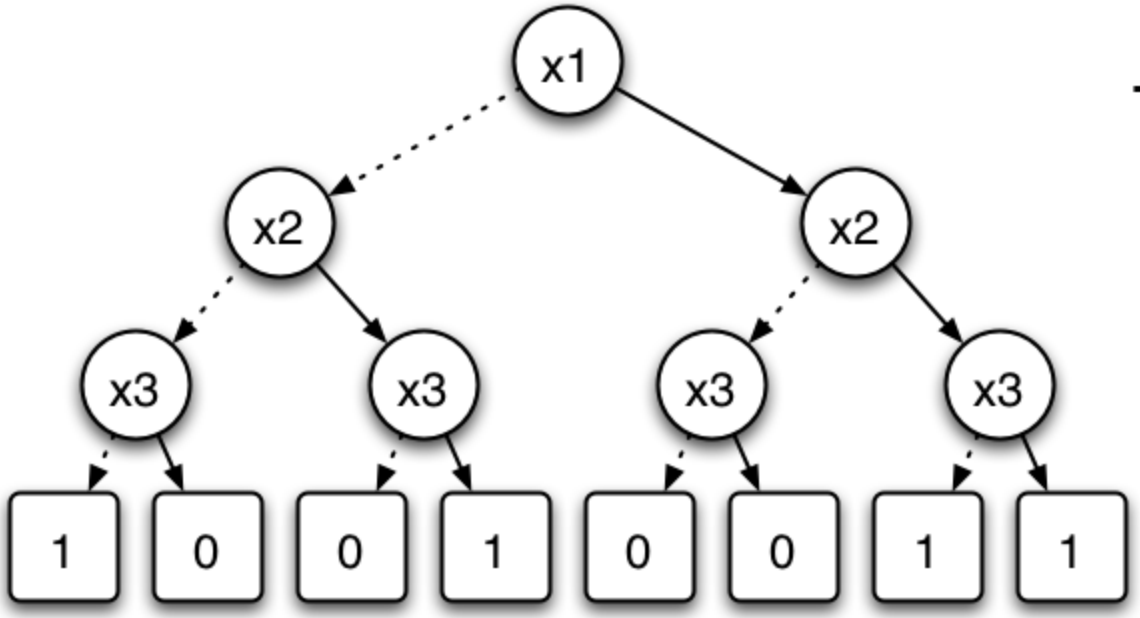
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Binary Decision Diagram (BDD)

- Dá sa ňou reprezentovať ľubovoľná Booleovská funkcia
  - Alternatíva pre pravdivostnú tabuľku, vektor alebo Karnaughovu mapu

$$f(x_1, x_2, x_3) = (\neg x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (x_1 \wedge x_2) \vee (x_2 \wedge x_3)$$

$$\bar{x}_1 \bar{x}_2 \bar{x}_3 + x_1 x_2 + x_2 x_3$$ (alternatívny zápis)



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

vektor

10010011

Zostrojenie nového BDD

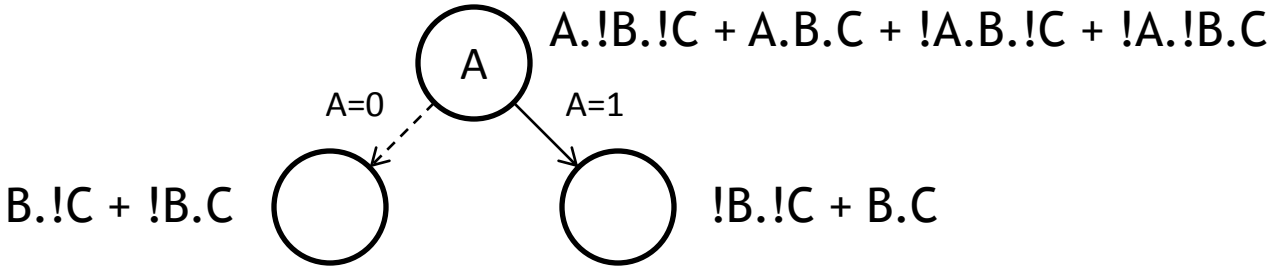
- Nový BDD sa môže zostrojiť relatívne jednoducho z ostatných spôsobov opisu Booleovskej funkcie
  - Bez ohľadu na to, akú konkrétnu Booleovskú funkciu opisujeme
- Dva prístupy:
  - Zhora nadol
    - postupnou dekompozíciou (rozkladom) podľa jednotlivých premenných
    - každá premenná predstavuje jednu úroveň v BDD
  - Zdola nahor
    - Postupným skladaním výstupov
    - Tiež každá premenná je jedna úroveň v BDD

Zostrojenie nového BDD – zhora nadol

- Shannonova dekompozícia (Shannon decomposition alebo Shannon extension)

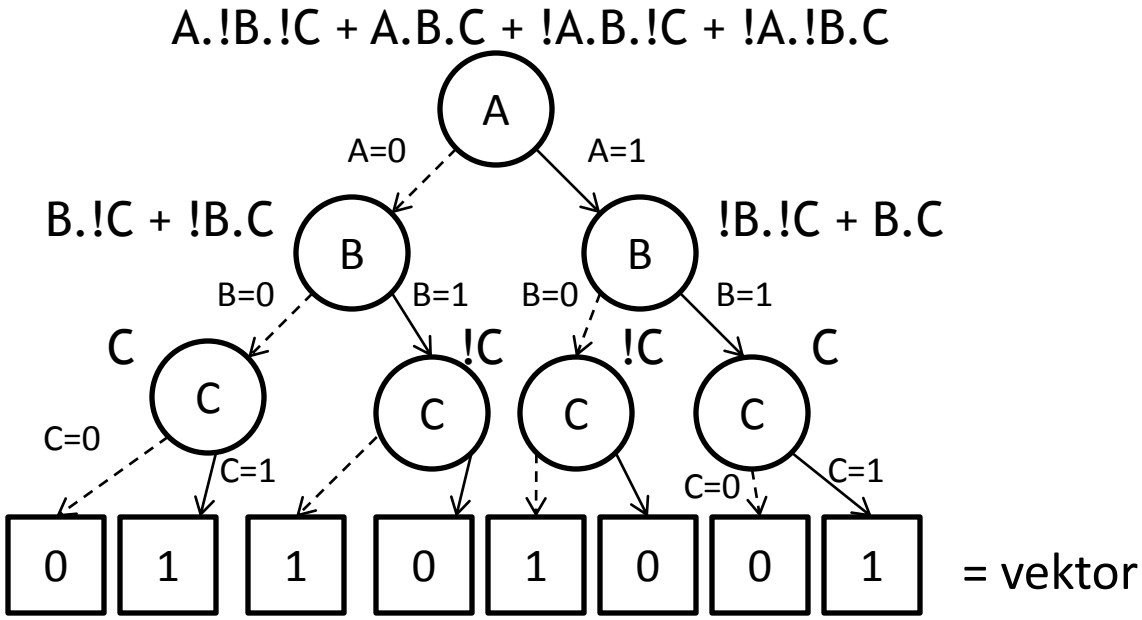
$$f(X_1, X_2, \ldots, X_n) = X_1 \cdot f(1, X_2, \ldots, X_n) + X_1' \cdot f(0, X_2, \ldots, X_n)$$

- Ak vytvárame BDD z výrazu v tvare DNF, napr.:
  - Y = A.!B.!C + A.B.C + !A.B.!C + !A.!B.C
  1. Vytvoríme koreň, ktorý reprezentuje celý výraz
  2. Vyberieme si jednu premennú, napr. A (čiže koreň je riadený premennou A)
  3. Y = A .(!B.!C + B.C) + !A .(B.!C + !B.C)
  4. Vytvoríme potomkov a vložíme zostatkové časti do nich
  5. Opakujeme kroky 2 až 4 pre každú premennú (t.j. N-krát, kde N je počet vstupných premenných Booleovskej funkcie)



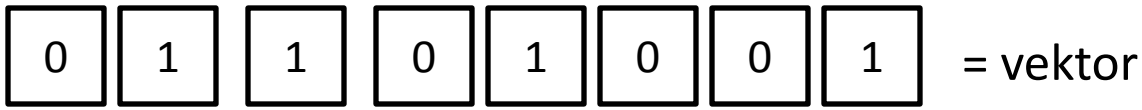
Zostrojenie nového BDD – zhora nadol

- Takže vyberieme zase nejakú premennú (napr. B)
- A pre všetky uzly v novej úrovni urobíme opäť Shannonovu dekompozíciu podľa premennej B
- B.!C + !B.C sa rozdelí na B.(!C) + !B.(C)
- !B.!C + B.C sa rozdelí na B.(C) + !B.(!C)
- Nakoniec predstavuje uzol len poslednú premennú alebo jej negovaný tvar



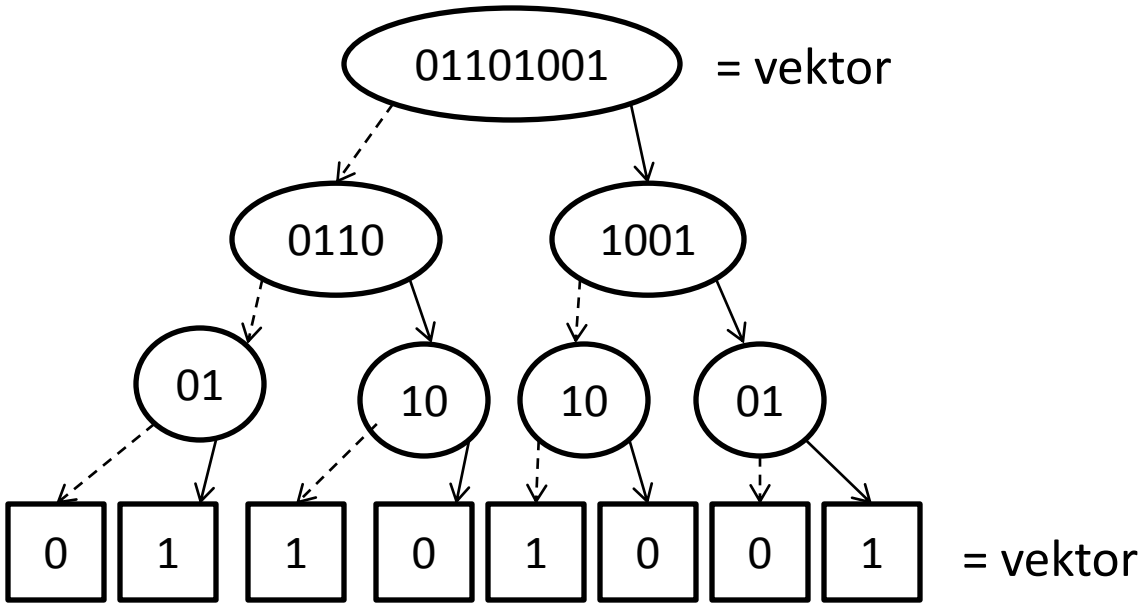
Zostrojenie nového BDD – zdola nahor

- Videli sme, že výsledný riadok s nulami a jednotkami je totožný s vektorom
- Preto ak máme zostrojiť BDD z Booleovskej funkcie opísanej formou vektora (alebo pravdivostnej tabuľky), môžeme tak urobiť prístupom zdola nahor
- Majme napr. vektor 01101001
- Najprv vytvoríme koncové uzly (listy) s týmito hodnotami v rovnakom poradí
  - Vytvoríme ako pole (koncových) uzlov



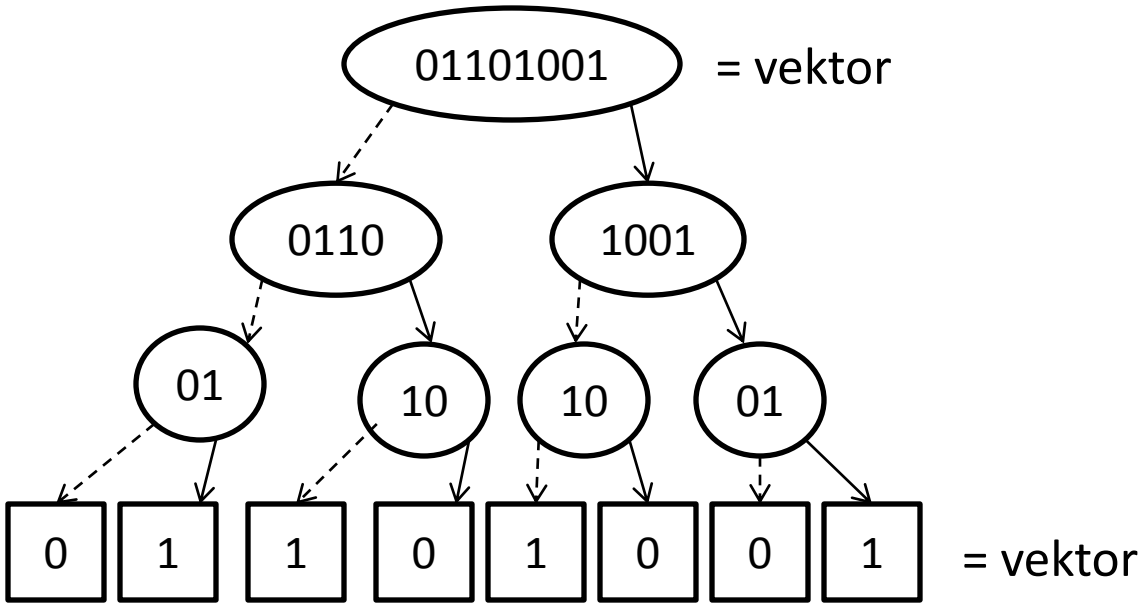
Zostrojenie nového BDD – zdola nahor

- Vezmeme vždy dvojicu susediacich uzlov a spojíme ich dokopy pridaním rodiča
- Rodič reprezentuje zlúčenú kombináciu hodnôt potomkov
- Opakujeme
- Poradie premenných je fixne dané podľa poradia vektora



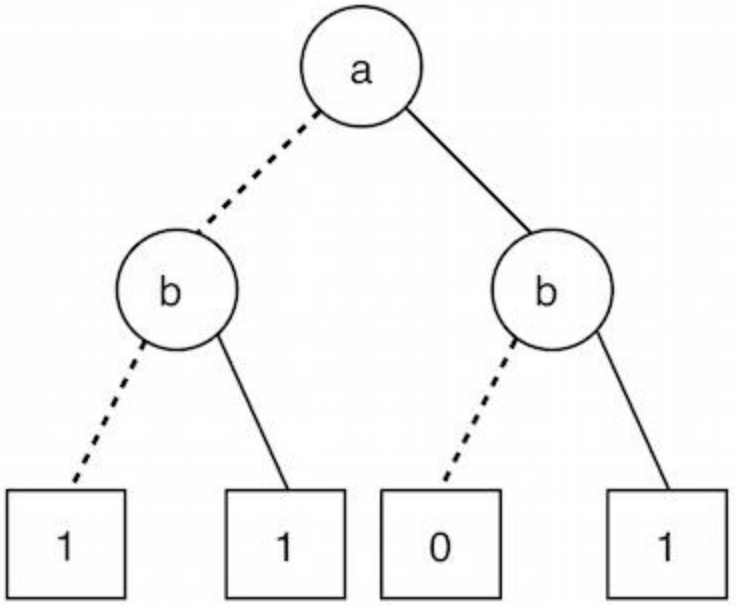
Zostrojenie nového BDD - zhora nadol

- Vektor → BDD
  - možné tiež realizovať prístupom zhora nadol
- Vektor delíme vždy na polovice
- Poradie je fixne dané, tak ako vo vektore
- Napr. vektor 01101001



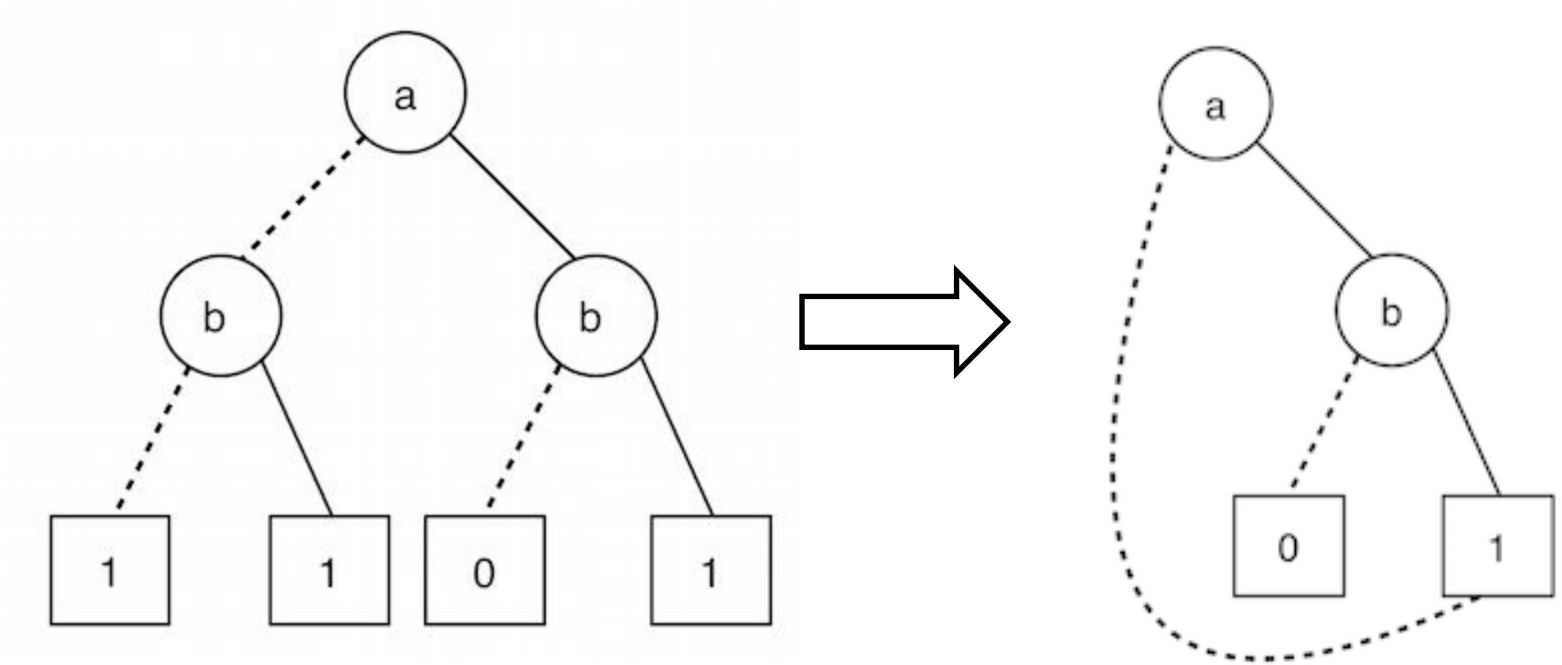
Problém Booleovských funkcií

- Pravdivostná tabuľka, vektor aj Karnaughova mapa sú reprezentácie, ktorých veľkosť je $\mathbf{2^N}$ pričom N je počet premenných Booleovskej funkcie
  - Exponenciálna zložitosť je problematická už pre N > 20
- **Rovnaký problém má aj BDD**
- Pridanie jednej premennej znamená pridanie ďalšej úrovne v BDD
- Jedná sa o úplný strom
- Neefektívne
- Ako to zlepšiť?
  - **Redukciou BDD**



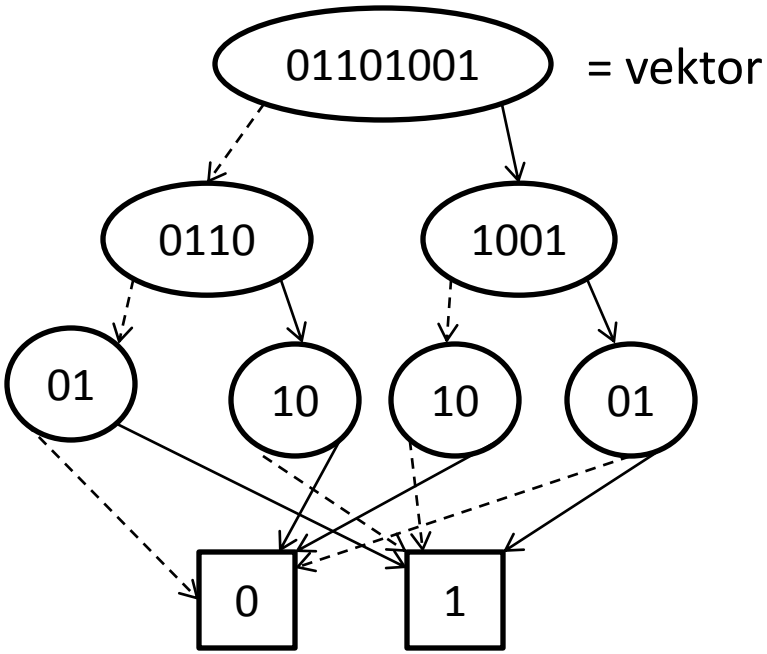
Redukcia BDD

- Počet uzlov by sme chceli minimalizovať
- Nie všetky uzly naozaj potrebujeme
- Odstránime redundantné (nadbytočné/zbytočné) uzly
- Redukcia exponenciálnej zložitosti až na lineárnu



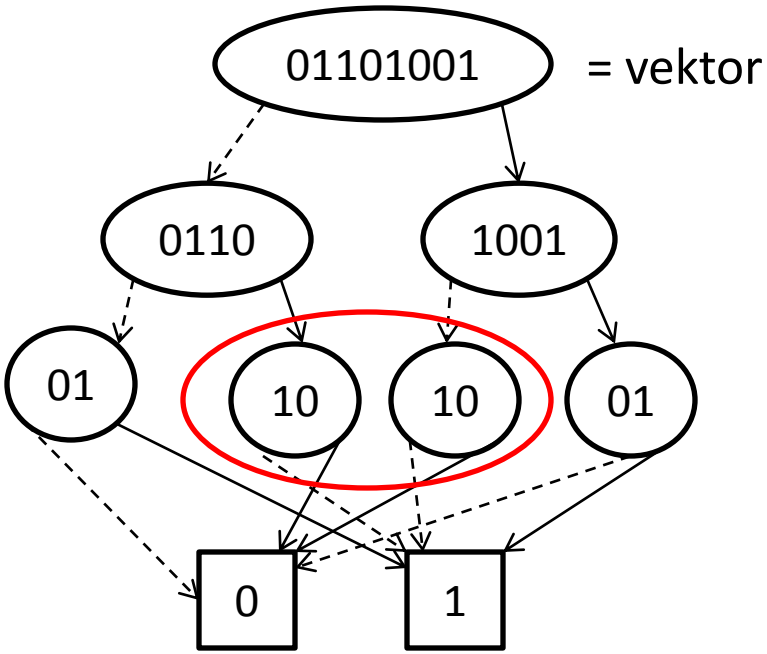
Zlúčenie koncových uzlov

- Konzový uzol (list) je vždy len 0 alebo 1 (apoň v prípade jedno-výstupových Booleovských funkcií)
- Takže nám stačia len dva
- Zlúčime všetky jednotky dokopy a všetky nuly dokopy, upravíme pointre



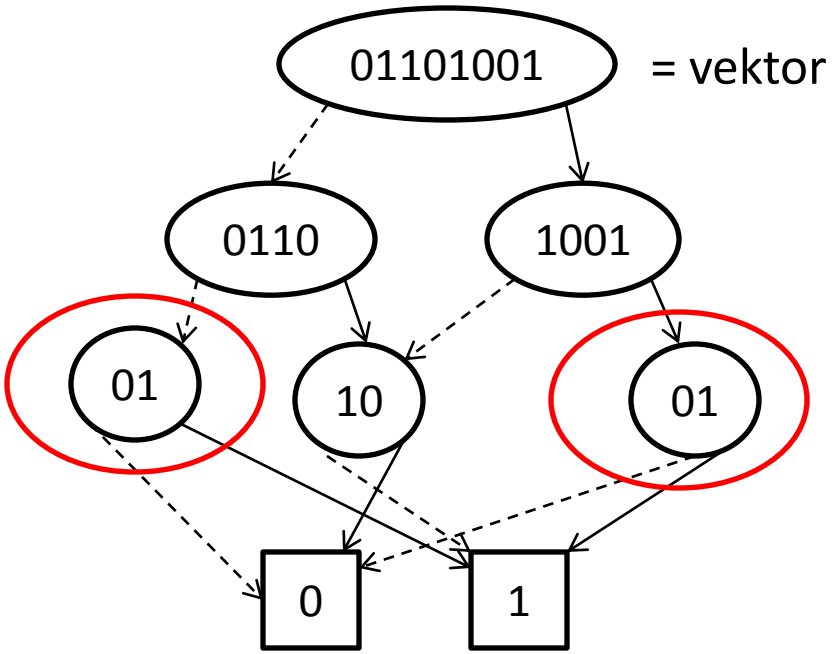
Zlúčenie vnútorných uzlov

- Ak je nejaký uzol nadbytočný, odstránime aj ten
- Ako zistíme, že uzol je nadbytočný?
  - Buď nemá žiadnu pridanú hodnotu
  - Alebo už taký istý uzol (s tou istou funkcionalitou) existuje



Zlúčenie vnútorných uzlov

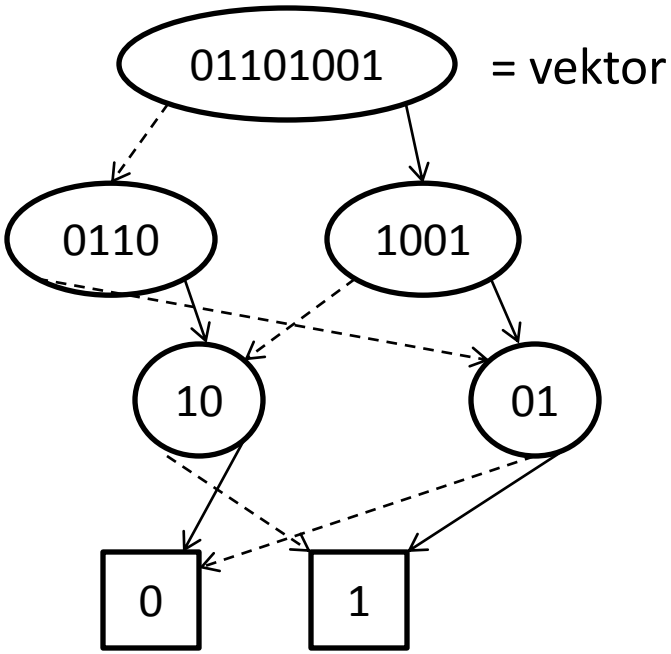
- Ak je nejaký uzol nadbytočný, odstránime aj ten
- Ako zistíme, že uzol je nadbytočný?
  - Buď nemá žiadnu pridanú hodnotu
  - Alebo už **taký istý uzol (s tou istou funkcionalitou)** existuje



Ešte nejaký?

Zlúčenie vnútorných uzlov

- Ak je nejaký uzol nadbytočný, odstránime aj ten
- Ako zistíme, že uzol je nadbytočný?
  - Buď nemá žiadnu pridanú hodnotu
  - Alebo už **taký istý uzol (s tou istou funkcionalitou)** existuje



Ešte nejaký?

Nie

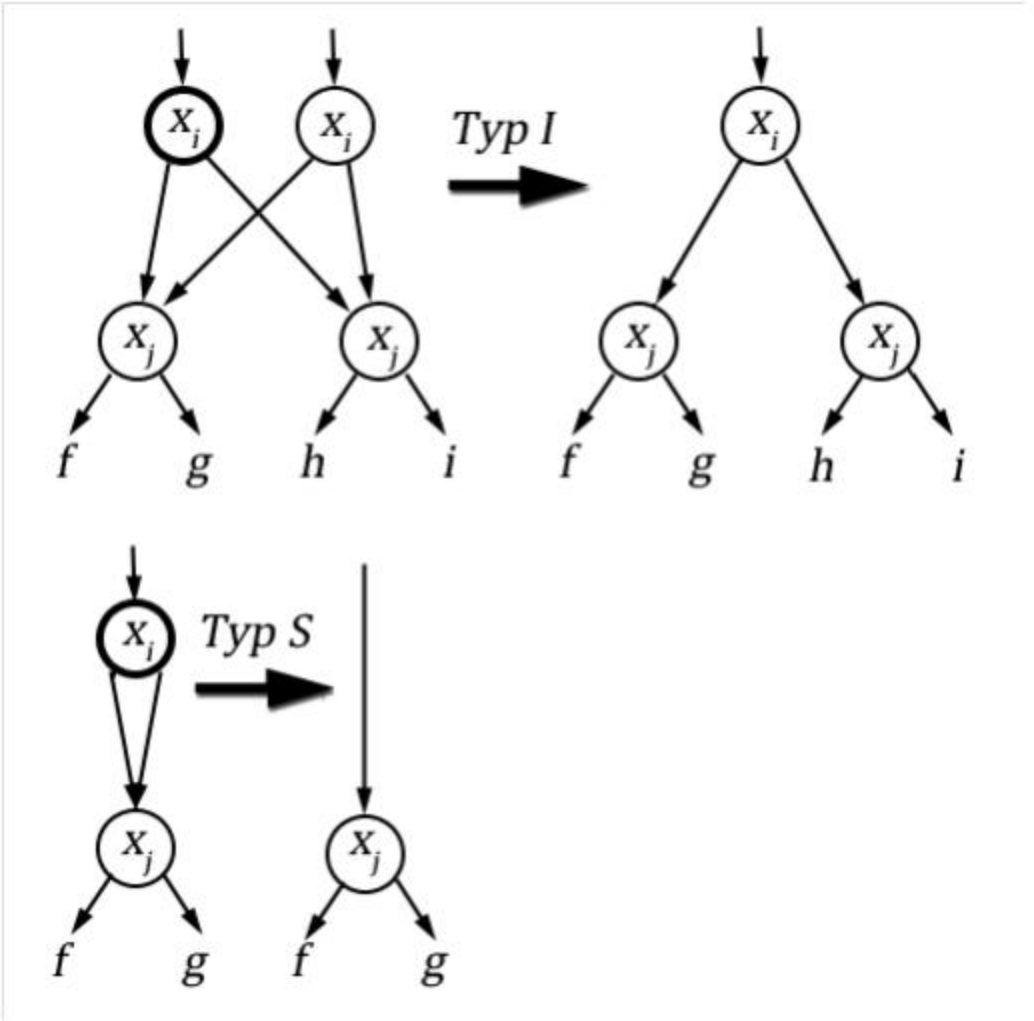
aspoň nie v tomto príklade...

Ako zistíme, že uzol je nadbytočný?

- Buď nemá žiadnu pridanú hodnotu
  - Kedy nemá žiadnu pridanú hodnoty?
    - Keď ľavý_potomok == pravý_potomok
      - Porovnávame ľavú polovicu funkcie s pravou
      - Porovnáme pointre na potomky
- Alebo už taký istý uzol (s tou istou funkcionalitou) existuje
  - Treba porovnať všetky dvojice uzlov medzi sebou a zistiť, či sú rovnaké
  - Stačí porovnávať len uzly v rámci jednej úrovne (jednej premennej)
  - Kedy sú 2 uzly rovnaké?
    - Buď máme v uzle napísaný opis funkcie, ktorú uzol realizuje
      - Porovnáme priamo opisy oboch uzlov
    - Alebo zistíme, či ľavý_potomok(uzol_1) == ľavý_potomok(uzol_2) a zároveň pravý_potomok(uzol_1) == pravý_potomok(uzol_2)
- **Pozor!!!** Porovnávanie pointrov je použiteľné len vtedy, ak už nižšia úroveň bola celá zredukovaná (t.j. redukcia smerom zdola nahor)

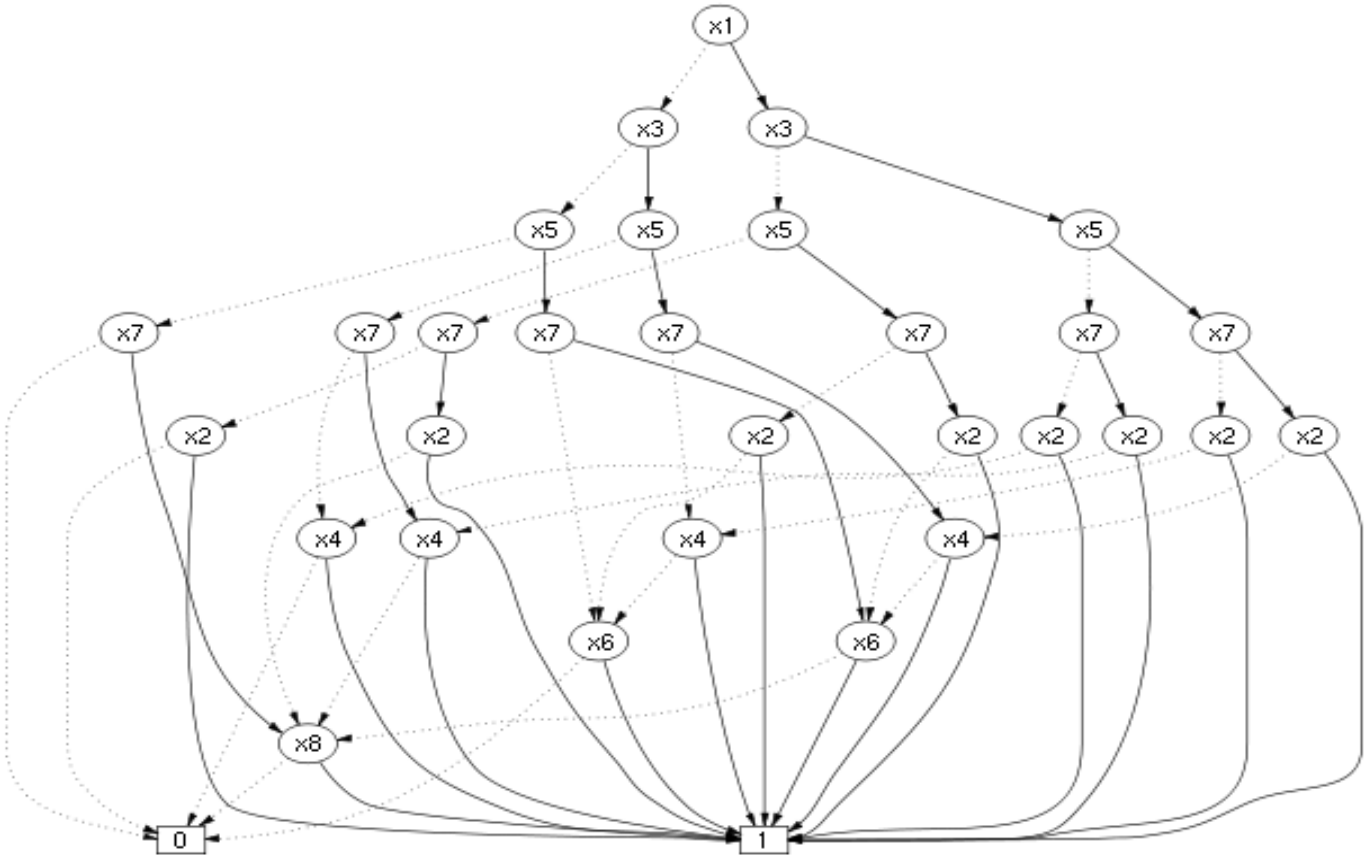
Pravidlá redukcie BDD

- Typ I – odstránenie nadbytočných uzlov porovnávaním dvojíc
- Typ S – odstránenie zbytočných uzlov porovnaním jeho potomkov



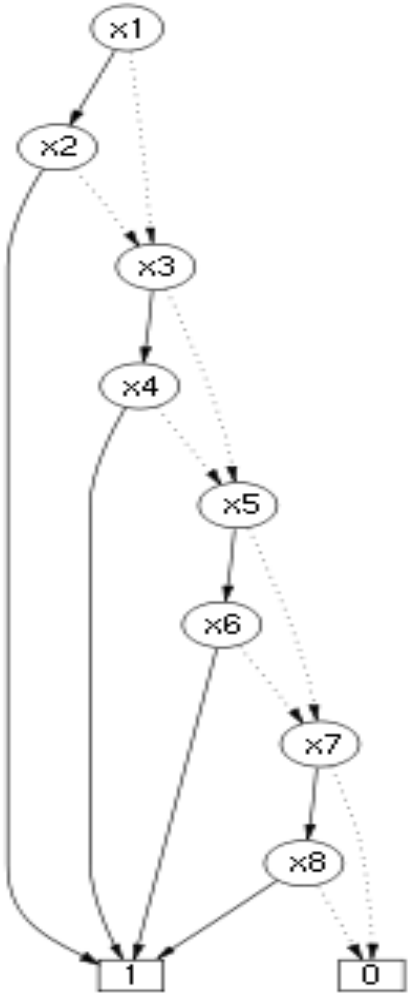
Redukovaný a zoradený = ROBDD

- Majme napr. funkciu Y = x1.x2 + x3.x4 + x5.x6 + x7.x8
- Zvolíme nejaké poradie premenných
- ROBDD môže vyzerať takto



Redukovaný a zoradený = ROBDD

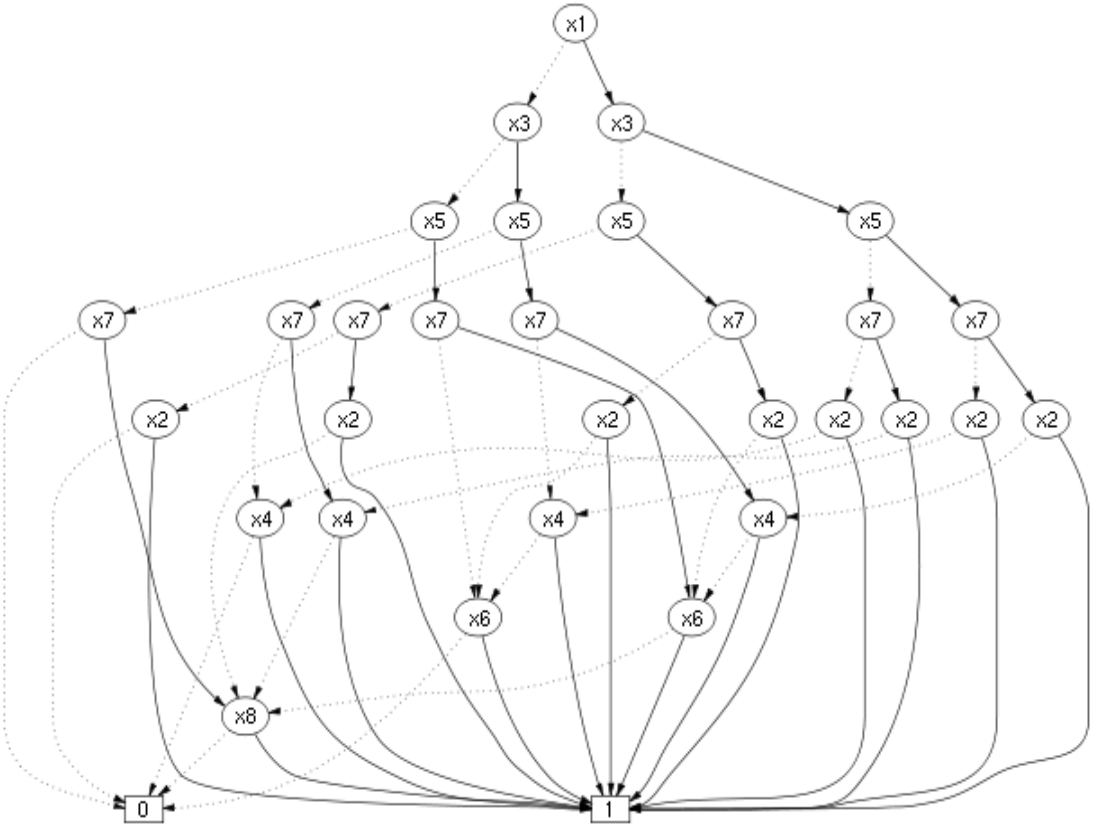
- Alebo aj takto ...

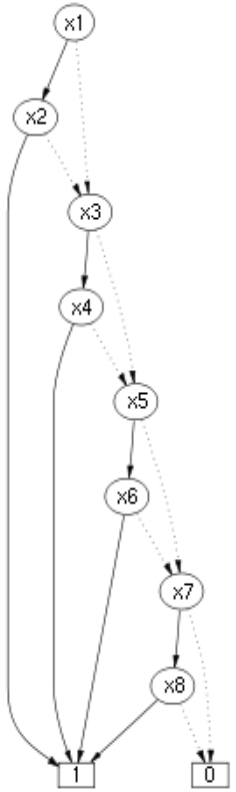


Ako je to možné?

Poradie premenných

- Poradie premenných dokáže výrazne ovplyvniť výslednú veľkosť ROBDD (t.j. počet uzlov)
- Prirodzene, chceme počet uzlov čo najmenší





Optimálne poradie premenných

- Použitím optimálneho poradia premenných dostaneme najmenší možný počet uzlov
- Ale ako zistíme optimálne poradie premenných?
- Väčšinou sa to robí systémom pokus-omyl
  - Vyskúšame nejaké poradie a pre toto poradie vytvoríme ROBDD
  - Zapamätáme si pre dané poradie počet uzlov
  - Zmeníme poradie premenných a znovu zostrojíme ROBDD
  - Porovnáme počet uzlov, ak sa zlepšil, pamätáme si toto poradie ako doteraz najlepšie
  - Opakujeme
    - Lenže koľko-krát?

Možnosti výberu poradia premenných

- Brute-force – vyskúšame všetky možné poradia premenných
  - Koľko ich je?
  - Máme N premenných, skúšame všetky permutácie, čiže N!
  - To je ale O(N!) – realistické tak pre N = 10 alebo menej
- Náhodne – náhodne vygenerujeme nejaké poradie (pomocou generátora náhodných čísel), X-krát opakujeme, X si zvolíme podľa toho, koľko času/úsilia sme ochotní tomu venovať
  - Problém opakovania toho istého poradia – plytvanie
- Lineárne – skúsime N možných poradí premenných, rotáciou premenných, napr. 12345, 23451, 34512, 45123, 51234
  - O(N) zložitosť, neopakujeme tie isté poradia
- Kvadratické – podobné ako lineárne, len s kvadratickou zložitosťou
- Pokročilejšie metódy (napr. hillclimbing, simulated annealing, genetic algorithms, ...)

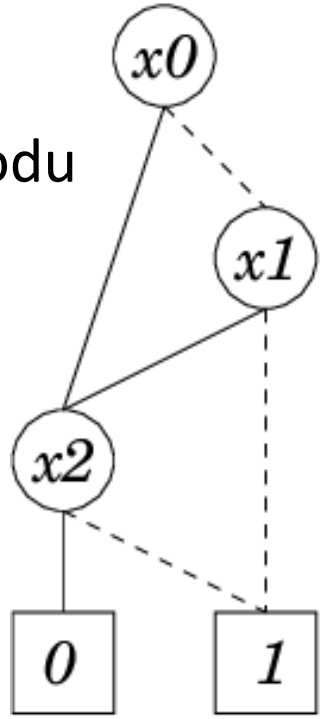
Multiplexorový strom

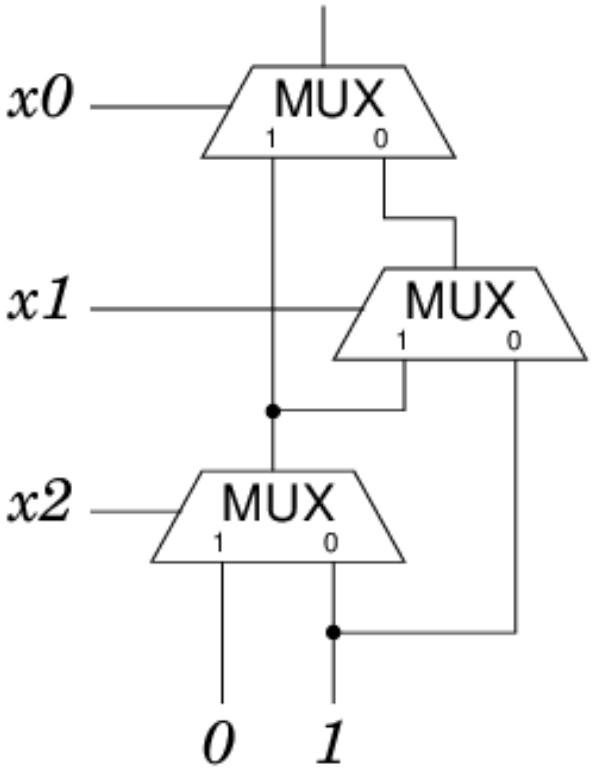
- Výsledný ROBDD môžeme použiť ako schému pre automatickú realizáciu (logickú syntézu) ľubovoľnej funkcie
- Výsledkom je kombinačný obvod (na čipe), ktorý dokáže realizovať zadanú Booleovsku funkciu
- Tento obvod sa nazýva multiplexorový strom
  - Strom multiplexorov (jednoduchých súčiastok)
- Mapovanie je 1:1
  - Každý uzol ROBDD sa transformuje na jeden 2-kanálový MUX
  - Hrany medzi uzlami budú vodiče spájajúce MUX-y (rovnakým zapojením)
- Riadiace vstupy do MUX-u sú jednotlivé premenné Booleovskej funkcie

Multiplexorový strom

- Čím menej uzlov má BDD, tým menej multiplexorov potrebujeme na realizáciu obvodu
  - Menšia plocha čipu, lacnejší, spoľahlivejší, s nižšou spotrebou energie

- Koreň BDD → výstup obvodu
- Uzol → MUX
- Hrana → vodič





Ďakujem za pozornosť