Zadanie 6 – Binárne rozhodovacie diagramy

Dmytro Shapovalov

id: 136278

Popis riešenia:

- V programe sa používajú 3 štruktúry:
  - BDD - štruktúra samotného binárneho rozhodovacieho diagramu, ktorá obsahuje počet uzlov, poradie premenných, ukazovateľ na výsledné uzly a ukazovateľ na uzol koreňa diagramu a hashovaciu tabuľku uzlov.
  - BDDNode - štruktúra jedného uzla diagramu s premennou, vetvou true, ak je premenná true, a vetvou false, ak je premenná false.
  - NodeHashTable - štruktúra hash tabuľky uzlov, v ktorej má každý uzol svoje vlastné id. Je vytvorená tak, aby algoritmus nevytváral duplicitné uzly. Vďaka tomu je diagram oveľa kompaktnejší

- Vysvetlenie fungovania hlavných funkcií:

- BDD_create - funkcia vytvorí štruktúru BDD, ktorá reprezentuje logickú funkciu ako binárny rozhodovací diagram - je to optimalizovaný rozhodovací strom, v ktorom každý uzol zodpovedá overeniu jednej premennej a cesty nadol vedú k hodnote true alebo false v závislosti od hodnôt premenných.

  Algoritmus funkcie:

  1. Najprv sa pridelí pamäť pre samotnú štruktúru BDD. Pomocou funkcie hashTableCreate sa inicializuje hashová tabuľka a pomocou funkcie createResultNodes sa inicializujú dva koncové uzly (výsledky): jeden pre 0 (false) a jeden pre 1 (true).

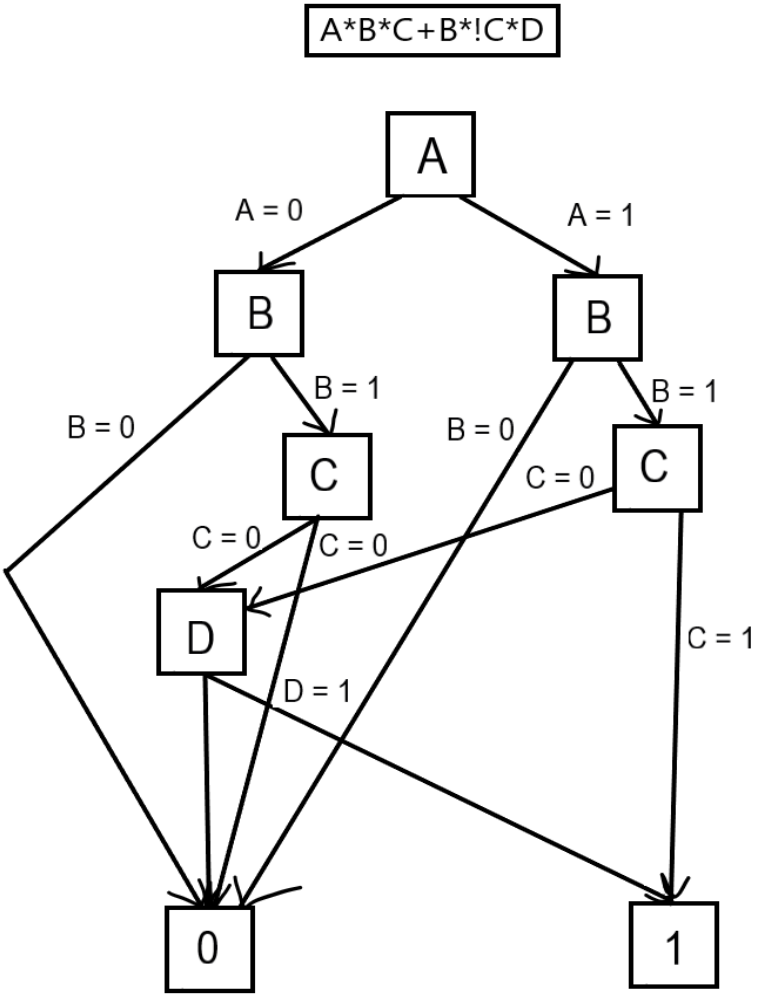
  2. Potom sa zavolá rekurzívna funkcia BDDbuild, ktorá vytvorí rozhodovací strom zhora nadol, pričom prechádza premennými v danom poradí. Na každej úrovni stromu sa vyberie aktuálna premenná a vytvoria sa dve „vetvy“: vetva, kde je premenná 1 (true), a vetva, kde je premenná 0 (false). Pre každú z týchto vetiev sa rekurzívne zavolá BDDbuild s aktualizovanou sadou hodnôt premenných.

  3. Vnútorné uzly sa vytvárajú prostredníctvom funkcie nodeCreate, ktorá: Kontroluje, či obe vetvy (true aj false) vedú do toho istého uzla. Ak áno, uzol sa nevytvorí. V opačnom prípade zahesluje kombináciu premenná + potomkovia, skontroluje tabuľku na duplicity, a len ak žiadne duplicity nie sú, vytvorí nový uzol.

  4. Ak rekurzia dosiahla koniec zoznamu premenných, zavolá sa funkcia leafDecide, ktorá: rozoberie logickú funkciu na termíny. Skontroluje, či je aspoň jeden z termov vykonaný pre aktuálne hodnoty premenných, ak áno, vráti uzol true (resultTrue), v opačnom prípade vráti uzol false (resultFalse).

5. Po zostavení stromu sa koreň BDD uloží do poľa root a počet uzlov sa zapíše do poľa nodeCount.

Príklad konštrukcie BDD pre bfunkciu A*B*C+B*!C*D:

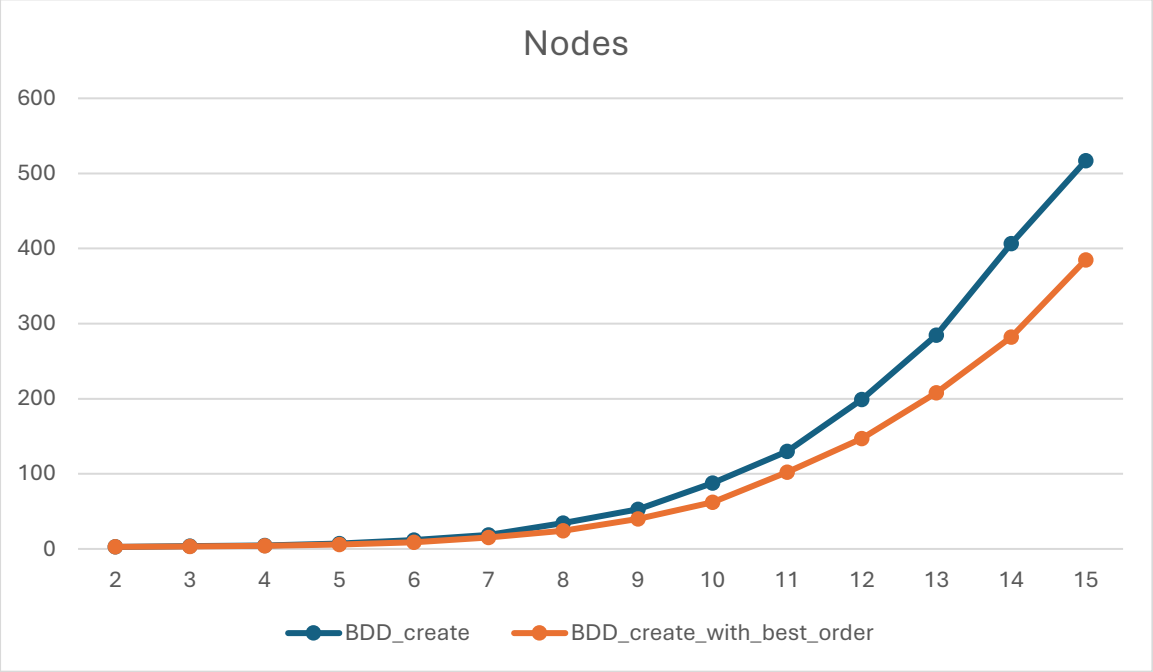


- BDD_create_with_best_order - funkcia vytvorí optimálny BDD pre danú bfunkciu tak, že porovnaním rôznych možností vyhľadá najefektívnejšie poradie premenných. Funkcia vykoná zadaný počet cyklov, pričom v každej iterácii vygeneruje nové náhodné poradie premenných pomocou funkcie orderCreate, vytvorí BDD s týmto poradím pomocou BDD_create a potom porovná počet uzlov (nodeCount) v aktuálnom BDD s predchádzajúcim najlepším výsledkom. Nakoniec vráti najlepší BDD s najnižším počtom uzlov.
- BDD_use - funkcia vypočíta výsledok boolovskej funkcie pre danú hodnotu premenných pomocou pripraveného BDD-diagramu. Začína od koreňa diagramu (bdd->root), v každom uzle vyberie vetvu (leafFalse alebo leafTrue) podľa hodnoty aktuálnej premennej. Pokračuje, kým nedosiahne koncový uzol (0 alebo 1). Na konci vráti hodnoty dosiahnutého koncového uzla.

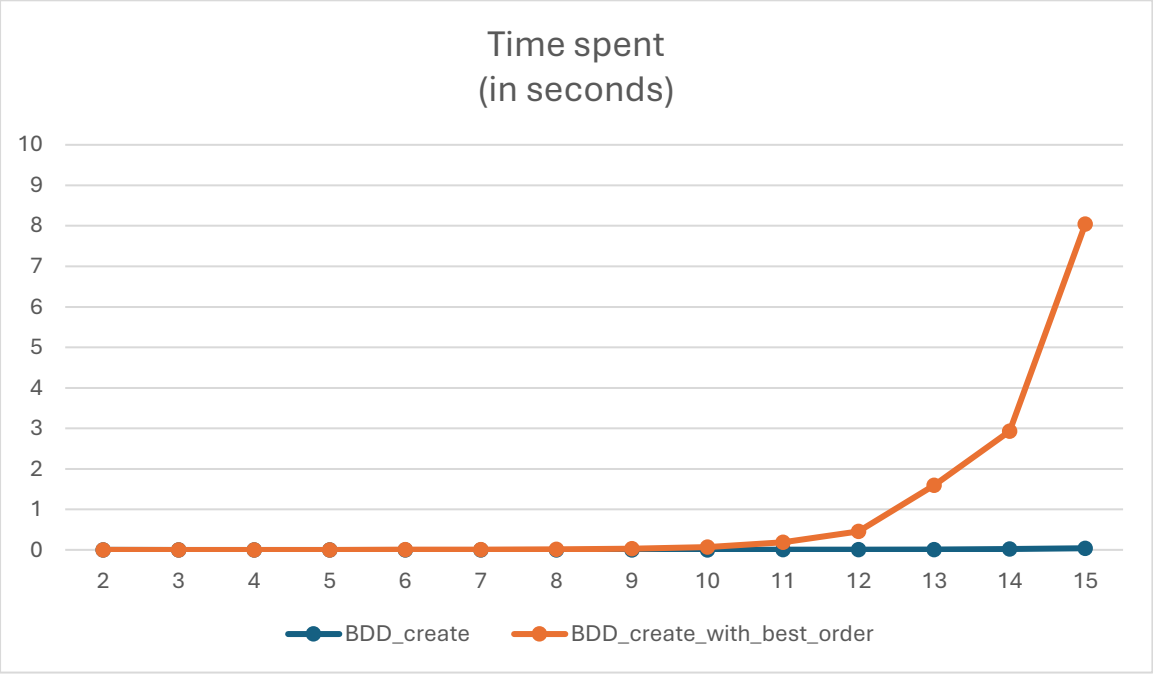
Testovanie:

- Kontrola správnosti BDD: na tento účel som vytvoril funkciu void resultsTest (), ktorá vytvorí 3 rôzne veľké BDD (5 premenných, 10 a 15) a kontroluje správnosť výsledkov funkcie BDD_use s rôznymi vstupnými údajmi pri použití. Všetky odpovede sa zhodujú, čo potvrdzuje, že implementácia štruktúry je správna .

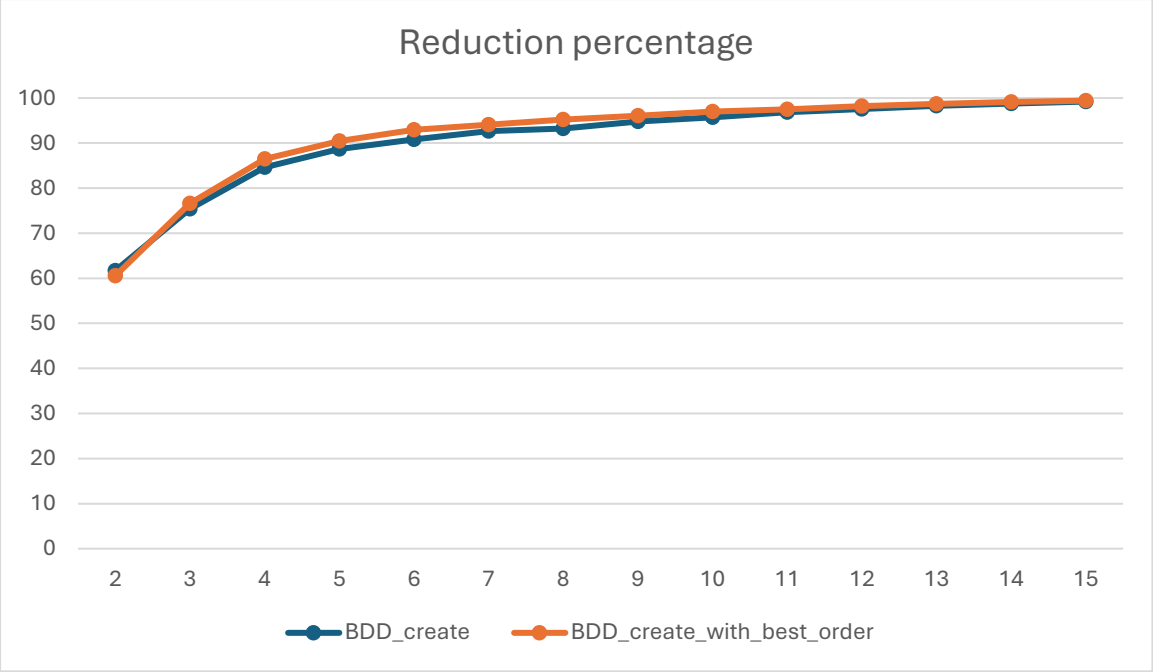
- Kontrola počtu uzlov vytvoreného BDD, času stráveného vytvorením a percenta redukcie pri použití funkcie BDD_create a BDD_create_with_best_order s rôznym počtom premenných (od 2 do 15). Pre každý počet premenných bolo vytvorených 100 dnf a zistená priemerná hodnota pre každý parameter:



Pre BDD_create je priestorova zložitosť približne O(1.5^n)
Pre BDD_create_with_best_order je priestorova zložitosť približne O(1.4^n)



Pre BDD_create je časová zložitosť približne O(2^n)
Pre BDD_create_with_best_order je časová zložitosť približne O(2.5^n)



Z grafu percenta redukcie vyplýva, že dostaneme niečo podobné logaritmickej krivke. To tiež potvrdzuje, že implementácia štruktúry je efektívna a správna.

Na základe týchto 3 grafov môžeme konštatovať, že funkcia BDD_create vykonáva svoju úlohu rýchlejšie, ale BDD_create_with_best_order je efektívnejšia, pretože vytvára kompaktnejší diagram.