Ádám Kemény
ID: 116749

DSA - zadanie 2 - Binárne rozhodovacie diagramy

Funkcionalita programu

Program má 6 hlavných classes a vstupný súbor Test.txt:

- **BDD_vstupy** uchováva input na vytvorenie BDD z Test.txt
- **BDD** obsahuje referencie na vrch BDD, veľkosť a počet premenných
- **DiagramManager** slúži na prácu s BDD
- **Node** je jeden prvok diagramu
- **Program** číta súbor a pracuje s ostatnými classmi, zabezpečuje výpis relevantných informácii o behu programu do konzoly
- **Vypis** obsahuje pomocné funkcie ako spočítanie Nodes v BDD

Najprv načítam vstup z Test.txt do BDD_vstupy, aby som s nimi mohol lepšie pracovať. Potom vytvorím BDD cez DiagramManager. To robím Shannonovou dekompozíciou.

Volám funkciu BDD_nextLayer, ktorá odstráni znak podla zadaného poradia a následne rozdelí DNF na pravú a ľavú. Ak je v podformuli DNF daný člen, tak ju zapíšem doprava. Ak je negovaný tak doľava, ak tam nie je, tak na obe strany. Ak je niektorá strana prázdna DNF, tak to znamená že tam bude node s hodnotou ‘0’. Následne odstránim prehladávaný znak z obidvoch strán. Ak až potom je DNF prázdna, tak nastavým danú stranu na ‘1’. Ak sa ľavá a pravá DNF rovná, tak vynechám daný znak, a idem rovno riešiť ďalší -> redukcia. Následne ak je nejaká strana null, tak na nej zavolám funkciu BDD_nextLayer rekurzívne.

Následne zavolám funkciu BDD_redukcia, ktorá porovná všetky Nodes s rovnakým znakom podla DNF. Ak sa ich DNF rovnajú, tak odstráni jednu z nich a preorganizuje diagram ->redukcia.

Vytvorím štruktúru BDD s potrebnými údajmi o grafe ako veľkosť a vrátim ju späť pre Program. Takto si vytvorím istý počet BDD s rôznym poradím vstupov, a ponechám si najviac redukovaný.

Potom použijem BDD_use pre každý vstup. BDD_use prechádza graf podla vstupu. 0 -> prehladávam vľavo, 1 -> prehladávam vpravo. Ak narazím na Node so znakom ‘1’ / ‘0’, tak ho vrátim. Následne porovnám výsledok BDD_use a očakávaným výsledkom z testovacieho súboru. Ak sa nezhodujú, tak vypíšem na konzolu: "Nastala chyba pri prehladávaní!".

Nakoniec vypíšem dôležité informácie o vygenerovanom bdd, počet Nodes, maximálny počet Nodes, redukcia v % a priemerná dĺžka vykonania BDD_use a BDD_create.

Ádám Kemény
ID: 116749

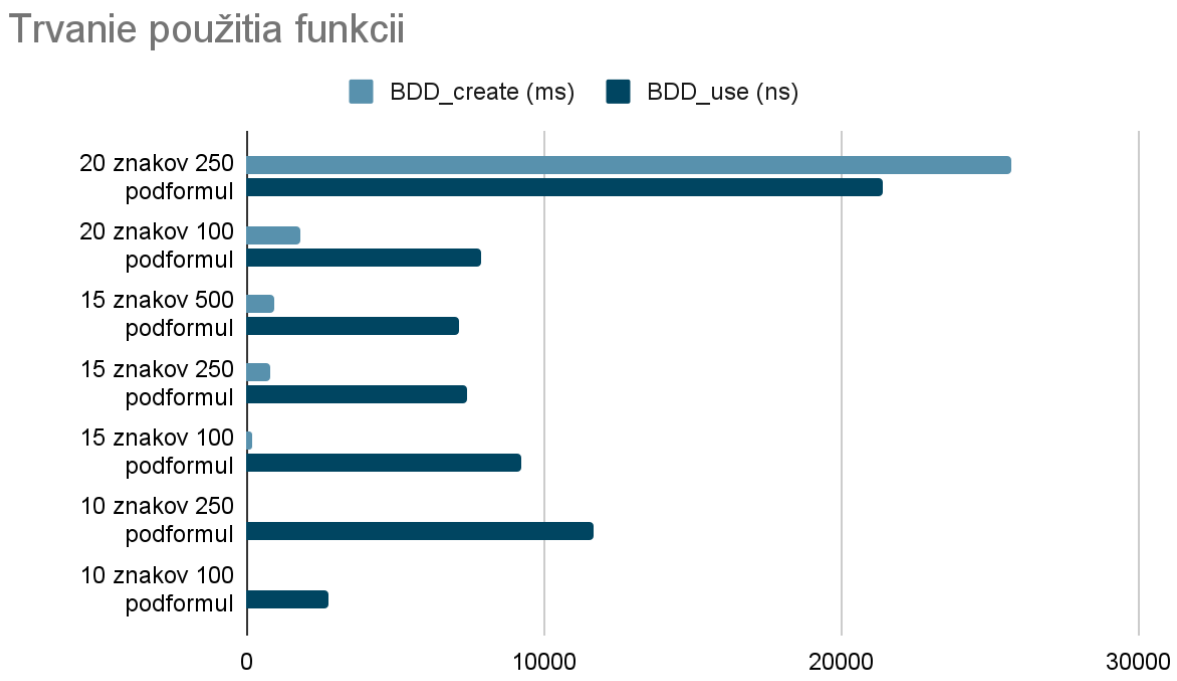
Funkcionalita testovacieho programu

Testovací program vytvorí podľa špecifikácii DNF formulu, vstupy a správne pravdivostné hodnoty k nim. Najprv si vygenerujem ***pocet_formul*** Stringov o veľkosti ***pocet_znakov***. Na každej pozícii je náhodne buď 0/1/2. To reprezentuje moju DNF formulu. Pozícia reprezentuje znak; [0] = a, [1] = b, [2] = c, … v podformule. Char na pozícii reprezentuje či znak *je (1) je negovaný (0) alebo nie je (2)* v danej podformule. Podformule sú potom oddelené s '+' pri zápise do súboru.

Následne náhodne generujem ***pocet_vstupov*** stringov o veľkosti ***pocet_znakov***. Sú naplnené 0/1 podľa pravdivostných hodnôt znaku na pozícii náhodne. Vstupy porovnám s DNF vygenerovanou predtým, a ak nájdem podformulu, kde sa všetky znaky zhodujú alebo tie kde sa nezhodujú majú v DNF na danom mieste '2', tak im priradím pravdivostnú hodnotu '1' Následne vypíšem do súboru Test.txt .

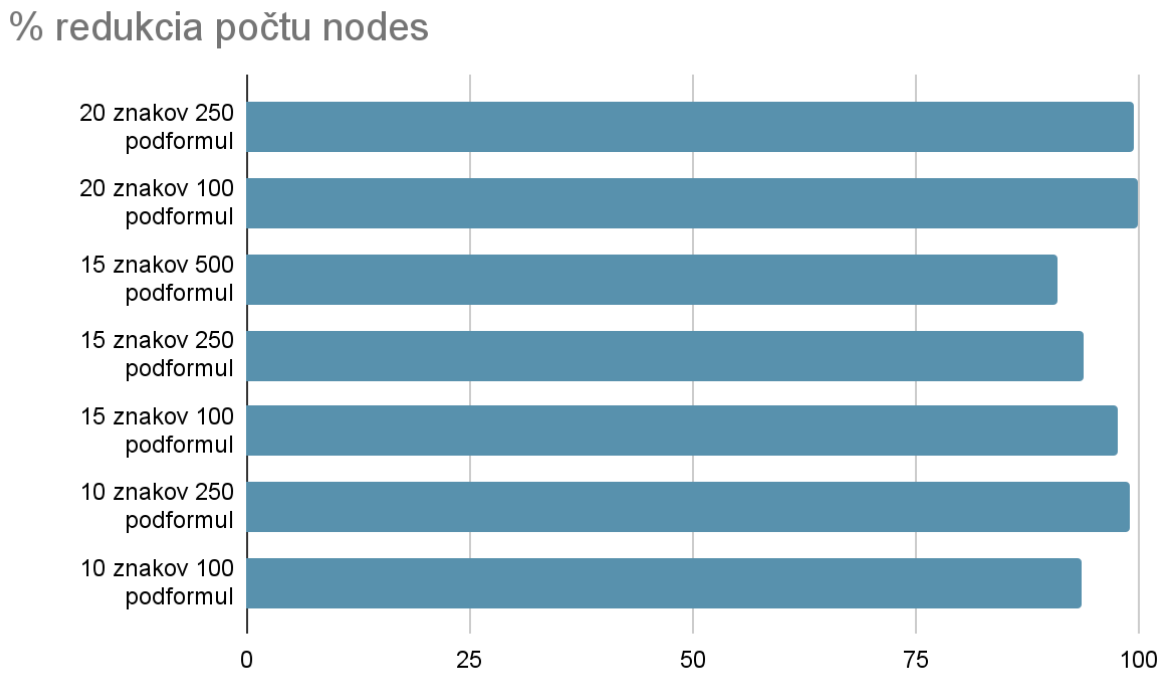
Nekontrolujem DNF any vyhľadávané postupnosti, takže sa môžu opakovať tie isté. To nie je pre môj program problém.

Efektivita programu



Čas potrebný na použitie daných funkcii v závislosti od počtu rôznych znakov a podformule DNF formuly. Pri dlhších DNF formulách a pri väčšom počte znakov sa čas potrebný na obe funkcie predlžuje. BDD_use lineárne, BDD_create podľa 2^2.

Ádám Kemény
ID: 116749



Pre lepšiu redukciu vygenerujem viacero BDD, a ponechávam ten, ktorý má najmenší počet Nodes.

zložitosť

Teoretická časová zložitosť programu je $O(2^n)$, kde n = počet rôznych znakov v DNF, keďže to je počet Node čo by trebalo vytvárať pre úplný strom. Reálna zložitosť však bude omnoho menšia v závislosti od počtu podformúl DNF.