FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
SLOVENSKÁ TECHNICKÁ UNIVERZITA
Ilkovičova 2, 842 16 Bratislava 4

2020/2021

Datové štruktúry a algoritmy

Zadanie č.3

Cvičiaci: Mgr. Martin Sabo, PhD.
Čas cvičení: Utorok 18:00 – 19:50

Vypracovala: Monika Zjavková
AIS ID: 105345

Obsah

1. Úvod ... 3

1.1. Binárne rozhodovacie stromy ... 3

1.2. Časová a priestorová zložitosť ... 3

2. Vytvorenie a používanie BDD ... 3

2.1. Vytváranie BDD ... 3

2.2. Používanie BDD ... 4

3. Redukovanie ... 5

3.1. Prepojenie smerníkov ... 5

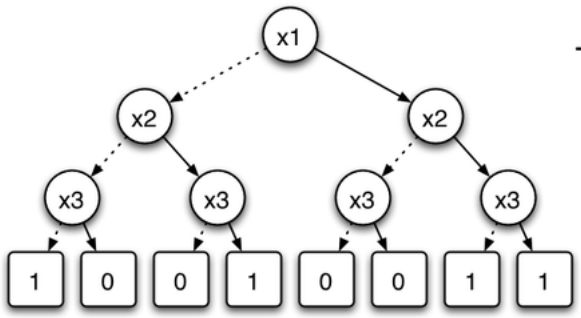
3.2. Vymazanie nadbytočných uzlov ... 6

4. Testovanie ... 7

1. Úvod

1.1. Binárne rozhodovacie stromy

Moja implementácia pracuje s vektormi, ktoré predstavujú výslednú hodnotu 0 alebo 1 po operácii rôzneho počtu premenných, čiže v pravdivostnej tabuľke posledný stĺpec.



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

Údaje sú uložené v štruktúrach. Jedna štruktúra slúži pre reprezentáciu uzla v strome, okrem reťazca, zobrazujúceho časť vektora, obsahujú smerníky na ľavý a pravý uzol. V listoch stromu sú oba nastavené na NULL. Všetky uzly tiež obsahujú smerník na rodiča, aby bolo možné pristupovať k súrodencom pri redukcii stromu a integer reduced, ktorý je na začiatku nastavený na nula. Po redukcii sa prepíše na jedna, aby program vedel, ktoré uzly sú už zredukované a netreba ich prechádzať.

Ďalšia štruktúra reprezentuje binárny rozhodovací strom. Nachádza sa v nej smerník na koreň stromu, počet premien, ktoré má strom zobrazovať a počet uzlov.

```
typedef struct Node{
    char *value;
    int reduced;
    struct Node *left;
    struct Node *right;
    struct Node *parent;
}Node;
```

```
typedef struct BDD{
    int varCount;
    int nodeCount;
    struct Node *root;
}BDD;
```

1.2. Časová a priestorová zložitosť

Časová zložitosť pri use v rozhodovacom binárnom strome je rovnaká ako vo vyváženom strome, keďže každý uzol ma práve dvoch potomkov. Zložitosť je teda O(logn). Priestorová zložitosť je O(n), pri čom n predstavuje počet uzlov v strome. Počet uzlov v strome sa vypočíta ako log2n.

Priestorová zložitosť sa pri reduce mení podľa toho, koľko sa tam nachádza identických uzlov. Pri vymazaní uzla, ktorý má rovnakú pravú a ľavú stranu, sa celkový počet uzlov zníži o dva. Keď sa na rovnakej úrovni v strome nachádzajú dva rovnaké uzly, zníži sa o jeden.

2. Vytvorenie a používanie BDD

2.1. Vytváranie BDD

Funkcia na vytváranie binárneho rozhodovacieho stromu využíva štruktúru BF, ktorá obsahuje vektor, podľa ktorého sa strom vytvorí. Alokuje sa miesto pre BDD štruktúru s počtom premien, ktorá sa vypočíta ako logaritmus dvojky z dĺžky, počtom uzlov v strome

zistenou z dvojnásobku dĺžky vektora a odčítania jednotky. Tiež obsahuje smerník na root stromu.

Strom sa vytvára rekurzívne smerom hora dole. Pridávanie prebieha podobne ako v obyčajnom binárnom strome. Na začiatku, keď vstupný uzol – koreň je NULL, vytvorí sa s plnou dĺžkou vektora.

```
if(node == NULL){
    node = createNode(value,NULL);
}

if(strlen(value)==1){
    return node;
}

node->left = createNode(left, node);
node->right = createNode(right,node);


insert(node->left,node->left->value);
insert(node->right,node->right->value);

return node;
```

Následne sa vektor rozdelí na dve polovice rovnakej dĺžky, z ktorých sa vytvorí pravý a ľavý uzol. Na vytvorenie nového uzla slúži pomocná funkcia createNode, kde sa alokuje nové miesto pre uzol, vloží sa vektor na vstupe. Okrem toho sa nastaví rodič, ktorým je pôvodný uzol. Pravý a ľavý potomok novovzniknutého uzla je nastavený na NULL. Funkcia potom vráti nový vytvorený uzol.

Funkcia pokračuje s rekurzívnym volaním, tento raz s ľavým a pravým uzlom ako parameter. Vektor sa vždy skráti na polovicu, keďže každý vznikne ako mocnina dvojky, vždy je možné rozdeliť ho na polovicu bez zvyšku.

V prípade, že už je dĺžka vektora jedna, funkcia skončí. Všetky listy teda obsahujú buď číslo jedna alebo nula, pri čom spoločne predstavujú vektor, ktorý bol na vstupe. Keď už je strom vytvorený uloží sa koreň do štruktúry BDD.

2.2. Používanie BDD

Binárny rozhodovací strom slúži na rozhodovanie, ktoré prebieha od koreňa ku listom, pri čom každý jeden uzol predstavuje jedno rozhodnutie, či sa posunie doprava – 1 alebo doľava – 0.

```
for (int i = 0; i < strlen(vstupy); i++) {
    if (strlen(node) == lenght) {
        if (vstupy[i] == '0') {
            node = node->left;
        } else if (vstupy[i] == '1') {
            node = node->right;
        }
    }
    lenght = lenght / 2;
}
if (strlen(node->value) == 1) {
    return node->value[0];
}
```

Kvôli redukcii je potrebné porovnávať aj dĺžku reťazca, ktorý sa nachádza v aktuálnom uzle. Predovšetkým kvôli tomu, že treba brať do úvahy prípad vymazania nadbytočného uzla, keď je jeho ľavá aj pravá strana rovnaká.

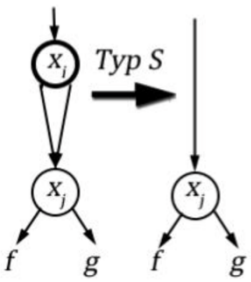
Aby nedošlo k posunutiu hodnôt, je tam vložená podmienka, ktorá ošetruje, či dĺžka hodnoty v uzle je polovička predchádzajúcej hodnoty. Ak by nebola, znamenalo by to, že uzol bol vymazaný a nie je potrebné urobiť žiadne rozhodnutie, keďže by mali rovnaký výsledok.

Rozhodnutie prebieha na základe zadanej hodnoty pre všetky premenné. To znamená, že pri dĺžke vektora osem, budú premenné tri a pri volaní funkcie use, musia mať zadanú hodnotu, ktorá môže byť v rozmedzí od 000, čo znamená, že všetky premenné sú nula a v binárnom rozhodovacom strome sa bude program pohybovať vždy doľava, až 111, čo je opačný prípad, kedy sú všetky premenné nastavené na 1 a strom sa posunie vždy po pravom uzle, kým nedôjde do listu.

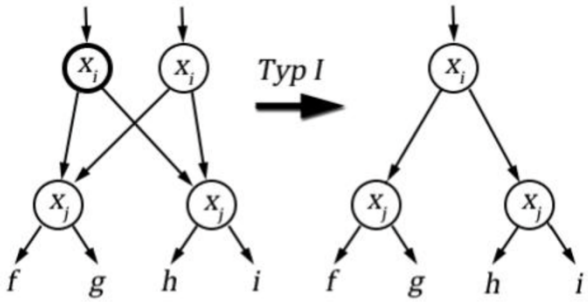
Keď program sa dostane do listu, vráti hodnotu v ňom zapísanú, ktorá predstavuje konečný výsledok – rozhodnutie pre danú postupnosť premien. Zároveň sa musí zhodovať výsledkom , ktorý je zadefinovaný vo vektore.

3. Redukovanie

Kvôli pamäťovej ja časovej náročnosti binárnych vyhľadávacích stromoch je efektívne strom po vytvorení redukovať. Keďže všetky uzly nie sú potrebné, je možné ich počet minimalizovať až o 90%. Redukciu môžeme spraviť v dvoch prípadoch. Keď je v uzle rovnaká ľavá a pravá strana je možné ho úplne odstrániť alebo prepojenie uzlov v prípade, že sa nachádzajú dva rovnaké v jednej úrovni stromu.



1.prípad



2.prípad

3.1. Prepojenie smerníkov

V prípade, že je uzol nadbytočný a v úrovni stromu sa nachádza s rovnakou funkcionalitou, môžeme vykonať redukciu zlúčením vnútorných uzlov v strome. Platí to aj pre všetky listy v strome, ktoré majú hodnotu 1 alebo 0 a zlúčia sa iba do dvoch listov.

```
if(strlen(node->value)==1){
    return;
}

if(node->parent!=NULL && node->reduced==0){
    findSibling(node,bdd->root,bdd);
}

reduce_redundant(node->left,bdd);
reduce_redundant(node->right,bdd);
```

Funkcia na odstránenie týchto nadbytočných uzlov využíva rekurziu, pre lepší prístup ku všetkým zlom. Prebieha dovtedy, kým dĺžka hodnoty v uzle nie je 1, čiže už sa nachádza v liste a nie je potrebné ani možné robiť ďalšiu redukciu.

Potom ak to nie je root, čiže má uzol svojho rodiča a ešte nebol redukovaný, zavolá sa funkcia na nájdenie všetkých uzlov na rovnakej úrovni. Pre nájdenie týchto uzlov je použitá ďalšia rekurzia, ktorá sa preruší až keď nájde uzol, ktorého hodnota má rovnakú dĺžku ako uzol, ktorý vstúpil do funkcie.

Keď už je nájdený druhý uzol z rovnakej úrovne, porovnávajú sa ich potomkovia. Aby nedošlo k chybe, v podmienke sa kontroluje, či druhý uzol má nejakých potomkov a či sa niektoré hodnoty rovnajú. Okrem hodnôt sa kontrolujú aj smerníky, aby nedošlo k prepísaniu.

Ak sú podmienky splnené, pre druhý uzol sa nastaví nový smerník, zníži sa počet uzlov v strome v štruktúre BDD a v zredukovanom uzle sa nastaví reduced na 1 ako označenie, že už uzol bol prejdený.

Následne sa všetky kroky zopakujú volaním funkcie rekurzívne s pravým a ľavým uzlom.

3.2. Vymazanie nadbytočných uzlov

Druhým krokom v redukcii je vymazanie nadbytočných uzlov. Uzol je nadbytočný v binárnom vyhľadávacom strome vtedy, keď jeho ľavý a pravý poduzol majú rovnakú hodnotu a teda v strome nie sú potrebné.

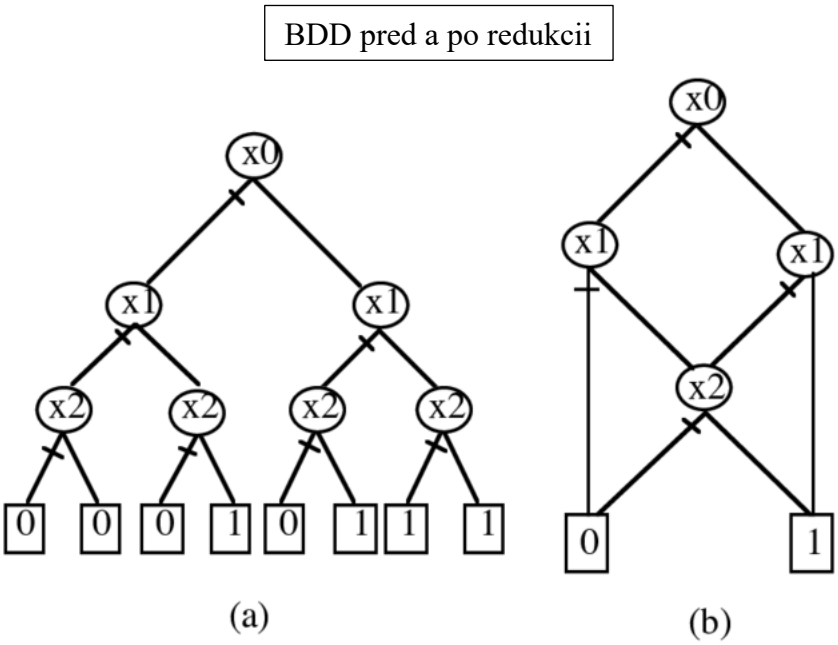
```
if(!strcmp(node->left->value,node->right->value)){
    if(node->parent == NULL){
        if(node->left->left!=NULL){
            node->left = node->left->left;
            bdd->nodeCount-=2;
        }
        if(node->left->right!=NULL){
            node->right = node->left->right;
            bdd->nodeCount-=2;
        }
    }
    else {
        if (node->parent->left == node) {
            node->left->parent = node->parent;
            node->parent->left = node->left;
            bdd->nodeCount-=2;
        } else {
            node->right->parent = node->parent;
            node->parent->right = node->right;
            bdd->nodeCount-=2;
        }
    }
}
```

Funkcia vyhľadáva tieto uzly pomocou rekurzie, ktorá prebieha, kým aktuálny uzol nie je NULL alebo je ho potomkovia nie sú NULL, aby nedošlo ku chybe, keď porovnávame potomkov.

Pre každý uzol sa potom porovná, či sa hodnota v ľavom a pravom uzle rovnajú. Ak áno, zistí sa, ktorý je to potomok, či ľavý alebo pravý. Následne sa na tú stranu namiesto neho zapíše jeden z jeho potomkov. Keďže majú rovnakú hodnotu, nie je dôležité ktorý.

Špecifický prípad, ktorý je potrebné ošetriť je root, keďže pracuje funkcia s rodičmi a root má nastaveného rodiča na NULL. V tomto prípade namiesto toho, aby sa vymazal samotný uzol a nahradil sa jedným z potomkov, sa prepíšu oba uzly, keďže root sa nemôže zmeniť. Na ľavú stranu sa zapíše ľavý poduzol jedného z potomkov a na pravú stranu pravý uzol potomka. Tiež nezáleží na tom, ktorého potomka si zvolíme, pretože sú obe strany rovnaké.

Následne sa od počtu celkových uzlov odčíta dva – počet uzlov, ktoré sa týmto ušetrilo. Funkcia sa volá rekurzívne znova tento raz s ľavým a pravým potomkom, čím prehľadáme celý strom a zastaví sa, až v listoch.



4. Testovanie

Testovanie funkcionality programu bolo pomocou porovnávania hodnôt pred a po redukcii na overenie, či aj po redukcii sa nájde správny výsledok vo rozhodovacom binárnom strome.

BDD štruktúry sa vytvárajú v cykle podľa zadaného počtu opakovania. Na začiatku sa náhodne vygeneruje vektor podľa, tiež so zadefinovanou dĺžkou na začiatku funkcie, podľa ktorého sa bude vytvárať strom. Potom sa vytvorí BF štruktúra, ktorá obsahuje vektor a jeho dĺžku. Táto štruktúra vstupuje do funkcie na vytvorenie BDD. Po vytvorení sa zaznamenáva počet uzlov, ktorý sa potom porovnáva s počtom po reduce.

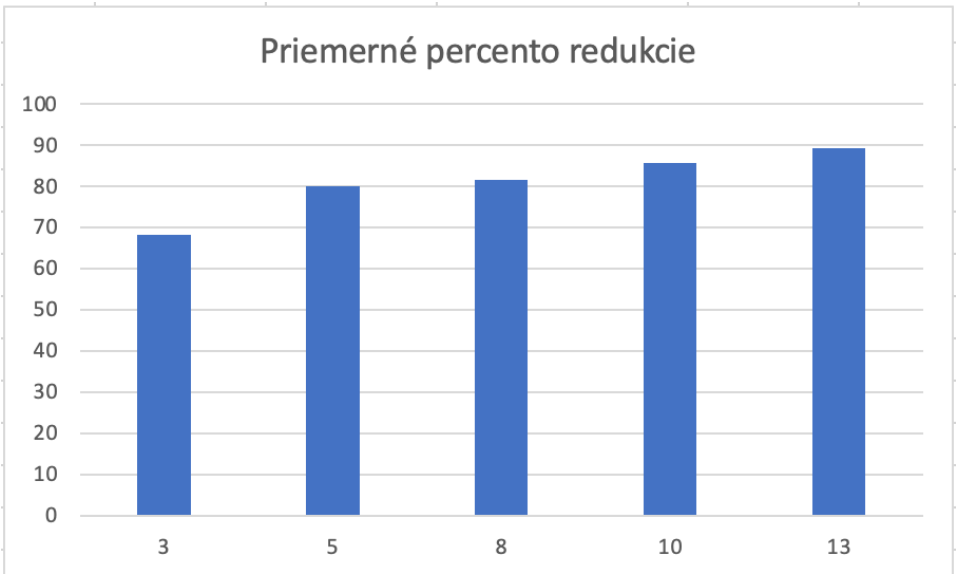
Keď už je strom vytvorený, testuje sa pre všetky kombinácie premien, ktoré môžu byť na vstupe a zaznamenávajú sa hodnoty na výstupe pre porovnanie.

Následne sa spustí reduce a porovnávajú sa výsledky po vyhľadávaní. Ak nastane chyba alebo rozdielny výsledok, vypíše sa pri ako vstupe, ak nenastane, nič z procesu sa nevypíše. Časy sú udávané v ms.

4.1. Porovnanie výsledkov

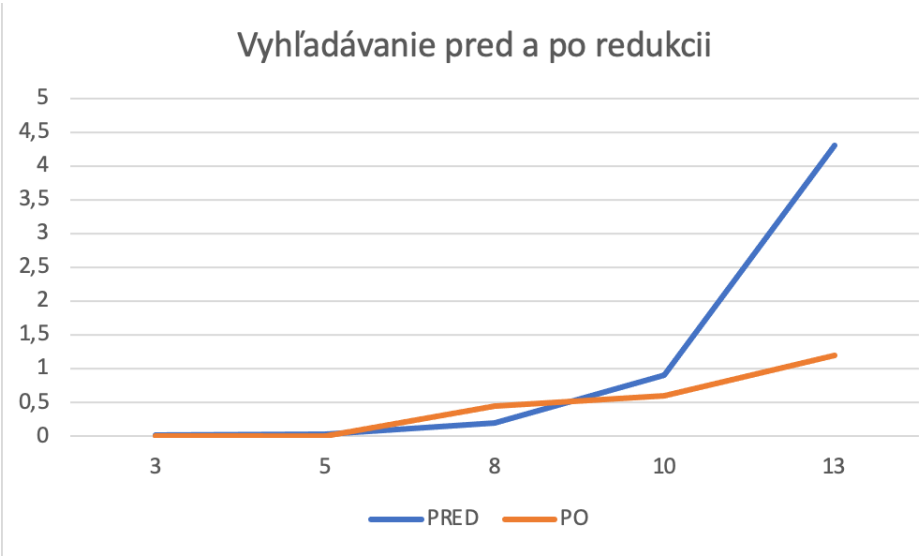
V programe boli testované scenáre podľa dvoch kritérií a to ako sa mení percento redukcie s počtom premenných a priemerný čas, ako dlho trvá pridávanie, redukcia a tiež priemerný čas, ktorý sa ušetrí po redukcii. Každý scenár bol testovaný pri:

- Počte premenných 3 a dĺžke vektora 8.
- Počte premenných 5 a dĺžke vektora 32.
- Počte premenných 8 a dĺžke vektora 256.
- Počte premenných 10 a dĺžke vektora 1024.
- Počte premenných 13 a dĺžke vektora 8192.

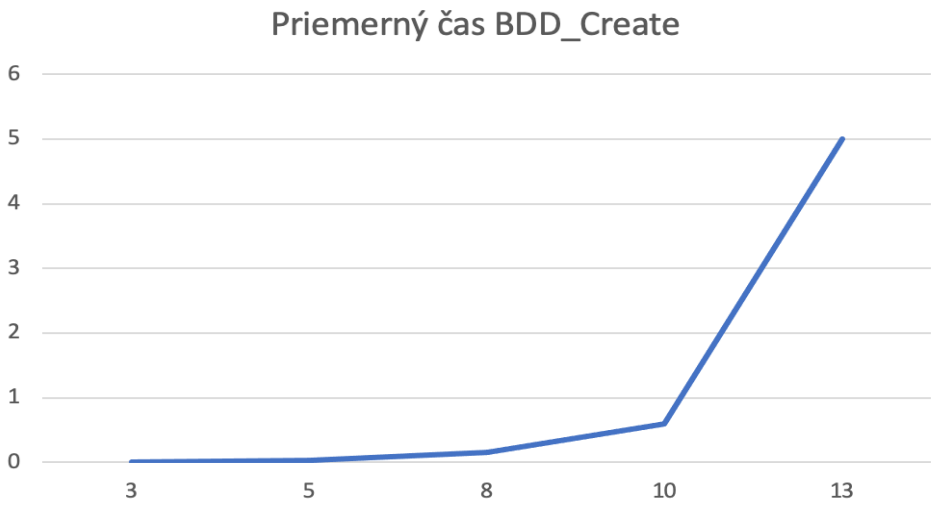


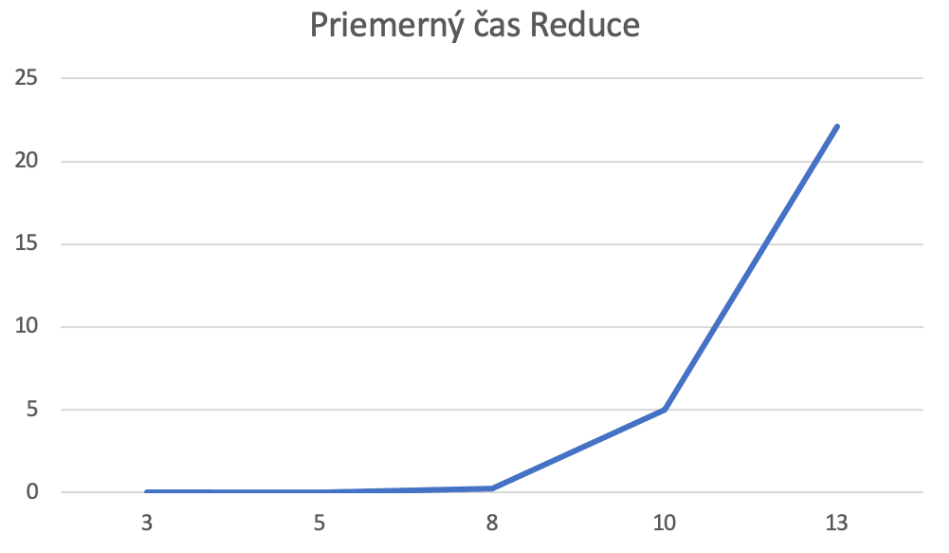
Percento úspešne zredukovaných uzlov sa zvyšuje s počtom premien a teda aj s dĺžkou vektora. Napríklad pri počte premien 3 je minimálny počet uzlov 2 – koreň stromu a dvaja rovnaký potomkova alebo 3, ak sú potomkovia odlišný, čo predstavuje 80 percent a väčšia redukcia už nie je možná.

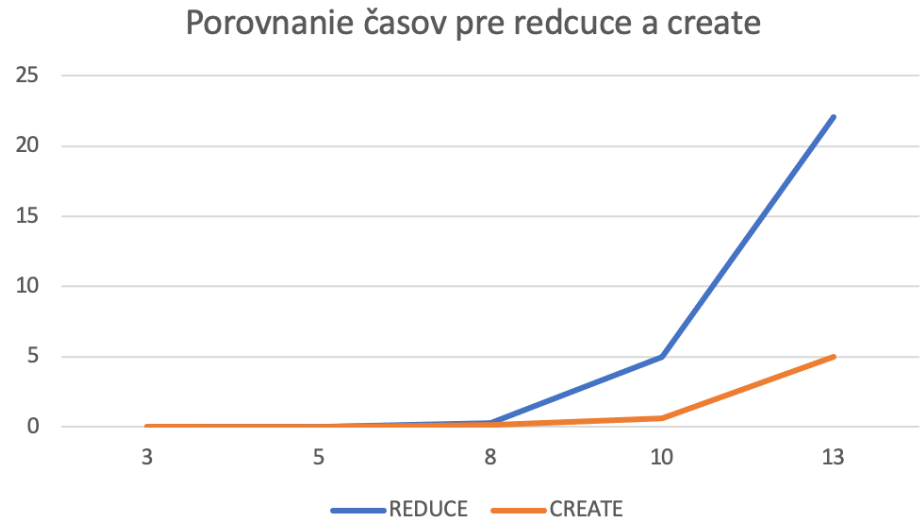
So stúpajúcim počtom premien je možné aj väčšie percento. Závisí to aj od kombinácie jednotiek a núl vo vektore, čím viac je rovnaký uzlov, tým je možná lepšia redukcia.



Redukcia umožnila urýchliť vyhľadávanie predovšetkým vďaka vymazania uzlov, ktoré mali rovnakú hodnotu v potomkoch. Ich vymazaním. Bolo ušetrenie jedno rozhodnutie a teda čas, ktorý. Bolo potrebné vynaložiť na presúvanie v strome.


Priemerný čas BDD_Create
6
5
4
3
2
1
0
3
5
8
10
13


Priemerný čas Reduce
25
20
15
10
5
0
3
5
8
10
13


Porovnanie časov pre redcuce a create
25
20
15
10
5
0
3
5
8
10
13
REDUCE
CREATE