**FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ**

**SLOVENSKÁ TECHNICKÁ UNIVERZITA**

Ilkovičova 2, 842 16 Bratislava 4

**2021/2022**

Dátové štruktúry a algoritmy

**Zadanie č.2**

**Cvičiaci: Mgr. Peter Lehoczký Vypracoval: Šimon Valíček**

**Čas cvičení: Štvrtok 13:30-14:40 AIS ID: 116320**

Šimon Valíček AIS ID: 105345

Obsah

**Binárny rozhodovací diagram ................................................................................................................3**

1. **Úvod.........................................................................................................................................3**
2. **Implementácia..........................................................................................................................3**
   1. *Zadefinovanie uzlu.............................................................................................................3*
   2. *Vkladanie...........................................................................................................................4*
   3. *Redukcia............................................................................................................................4*
   4. *Vyhľadávanie.....................................................................................................................5*

**Testovanie.............................................................................................................................................5**

1. **Úvod.........................................................................................................................................5**
   1. *Funckia na vyhodnotenie booleovského výrazu – implementácia.....................................5*
   2. *Generátor náhodných stringov - implementácia...............................................................6*
   3. *Tester a testovanie............................................................................................................6*
   4. *Miera redukcie..................................................................................................................7*

**Záver.....................................................................................................................................................8**

**Binárny rozhodovací diagram**

1. **Úvod**

V binárnom rozhodovacom diagrame sú údaje uložené do uzlov. Moja implementácia je riešená pomocou pridávania na základe postupnosti uzlov v binárnej sústave. Uzly pridávam rekurzívne a každou rekurziou sa redukuje daný booleovský výraz, až pokiaľ nezískame hodnotu 1 alebo 0 na každom konci diagramu. Štruktúra jedného uzlu diagramu pozostáva z booleovského výrazu pre funkciu, ktorá je typu string, poradie daného uzlu v rámci stromu typu int a odkazom na pravý uzol, ľavý uzol a uzol rodiča.

Moje riešenie som sa rozhodol implementovať v prostredí C++, avšak pomocou procedurálneho štýlu programovania. Pôvodne som mal v pláne použiť jazyk C, avšak nakoľko sme mohli použiť prevzatú funkciu na vyhodnocovanie booleovských výrazov a v C som žiadnu nenašiel, rozhodol som sa pre C++. To sa mi nakoniec nepodarilo nájsť ani pre C++, a tak som implementoval vlastnú, avšak beriem to pozitívne, nakoľko mi aspoň odpadla práca so stringami, ktorej bolo v tomto projekte viac než dosť. Vzhľadom na procedurálny spôsob riešenia danej problematiky som sa dopracoval k výsledku, že môj kód má takmer 1000 riadkov a jednotlivé funkcie sú natoľko dlhé, že ich mnohé v dokumentácii ani neuvádzam, nakoľko snímok obrazovky by bol v niektorých prípadoch až na dve strany vo Worde.

1. **Implementácia**
   1. **Zadefinovanie uzlu**

V úplnom úvode písania môjho projektu som si musel v prvom rade zadefinovať štruktúru uzlu. Tento proces som už viacmenej opísal v úvode tohto projektu.

Obrázok, na ktorom je text

Automaticky generovaný popis

* 1. **Vkladanie**

Opäť, ako som už raz spomínal v úvode, vkladanie do diagramu som zabezpečil pomocou binárnej pozície daného uzlu. Využil som pri tom poznatky z predmetu ADM, kde sme sa učili o prehľadávaní do hĺbky. Číslovanie uzlu začína od koreňa, ktorý má pozíciu 1. Pre každú ďalšiu pozíciu som previedol toto číslo do binárnej sústavy a odstránil som z nej prvý znak (0 alebo 1).

Pre pochopenie uvediem príklad. Číslo 2 je v binárnej sústave „10“. Odstránim z neho prvý znak a získam „0“, čo mi predpovedá, že pôjdem doľava. Pre číslo 3 je to „11“, po odstránení prvého znaku získam „1“, čo mi napovedá, že sa budem pohybovať v diagram smerom doprava.

Takto pokračujem ďalej, pričom číslovanie prebieha zľava doprava, odhora nadol. To znamená, že najprv prejdem všetkými uzlami na prvej úrovni a až potom postupujem na ďalšiu úroveň.

Ako som vyššie spomínal, využil som pri tom algoritmus BFS.

Čo sa týka prechádzania po vetvách, vstupom funkcie je reťazec, nachádzajúci sa na koreni diagramu. Ďalším argumentom funkcie je odkaz na koreň tohto diagramu. Ak sa koreň rovná nule, vytvorím uzol a funkcia vráti ukazovateľ na tento uzol, inak postupujem podľa toho, či je nasledujúca v poradí 0 alebo 1. Ak nula, najprv vypočítam hodnotu reťazcu pre písmeno na danom leveli (úrovni diagramu) volaním pomocnej funkcie. Takmer identickú funkciu som implementoval aj pre pravú vetvu diagramu. Návratovou hodnotou týchto funkcii je zredukovaný string. Ak sa výraz rovná nule, funkcia mi vráti „0“. Zároveň každým prechádzaním funkcie na vkladanie uzlov do stromu volám funkciu, ktorá zisťuje, či sa výraz rovná „1“. Funguje na princípe, že ak sa v nejakom booleovskom výraze nachádzajú iba písmenká, ktorých level sme už prehľadávaním prešli, funkcia vráti návratovú hodnotu „1“. V prípade, že zredukovaný výraz sa rovná „0“ alebo „1“, vrátim ukazovateľ na daný výraz a uzol nevytváram.

* 1. **Redukcia**

Dôležitá časť tohto projektu je redukcia. Tá v mojom prípade prebieha tak, že ak má nastať vloženie uzlu do stromu (už s redukovaným booleovským výrazom), tesne pred pridaním prehľadám diagram a zistím, či sa už daný výraz v diagrame nachádza. Ak nie, vrátim nulový ukazovateľ a uzol pridám na dané miesto v strome podľa poradia. Ak áno, návratová hodnota funkcie, ktorá mi zabezpečí prehľadávanie stromu je ukazovateľ na uzol s tým istým identickým booleovským výrazom, prípadne 1 alebo 0. Na prehľadávanie tohto stromu som implementoval dve funkcie, jednu pre pravú časť, druhú pre ľavú časť. Podľa toho, ktorá funkcia mi daný ukazovateľ vráti, priradím rodičovi uzlu (ktorý som išiel pôvodne vložiť) buď ľavý alebo pravý ukazovateľ. Funkcie sú implementované tak, aby mi vždy najviac jedna z nich vrátila ukazovateľ na daný zhodný reťazec v diagrame.

* 1. **Vyhľadávanie**

Vzhľadom na to, že vkladanie prvkov do diagramu mám implementované na tej istej báze ako vyhľadávanie, táto funkcia bola pre mňa jednoduchšia. Vstupom je string zložený z 0 a 1 oddelených bodkami, ktorý si vo funkcii rozdelím na jednotlivé písmenká a následne pomocou toho či je v poradí 0 alebo 1 volám buď funkciu, ktorá mi vráti ukazovateľ na ľavú vetvu, alebo funkciu, ktorá mi vráti ukazovateľ na pravú vetvu. Týmto istým spôsobom pokračujem až kým nevyčerpám všetky prvky v zadanom reťazci, alebo kým neprídem k nule alebo jednotke na konci vetvy.

**Testovanie**

1. **Úvod**

Testovanie môjho programu spočíva na báze dosadenia 100 rôznych booleovských výrazov do diagramu a zistenie výsledných hodnôt na daných vetvách pre všetky možné kombinácie vstupov 0 a 1. Súčasne rovnaké výrazy a vstupy dosádzam aj do pomocnej funkcie, ktorá mi vyhodnotí hodnotu pre booleovský výraz, na základe rozobratia stringu na jednotlivé sčítance a násobky. Túto funkciu som si sám implementoval

* 1. **Funkcia na vyhodnotenie booleovského výrazu - implementácia**

Vstupnými argumentmi funkcie sú string, obsahujúci testovaný booleovský výraz, ďalej string, obsahujúci poradie premenných pre booleovský výraz a následne string, obsahujúci kombináciu núl a jednotiek. Všetky tieto stringy si rozdelím nasledovne. Testovaný booleovský výraz rozložím najprv na polia stringov, obsahujúce sčítance. Jednoduchšie povedané, hľadám znamienka „+“, ktoré považujem za oddelovače. Každé takto vzniknuté pole ešte rozdelím podľa znamienok „.“, ktoré takisto považujem za oddelovače. Ďalej si rozdelím string, obsahujúci vstupné poradie premenných. Tieto písmenká zapíšem do poľa, pričom vytvorím druhé pola, obsahujúce tie isté písmenká avšak znegované. To robím jednoducho tak, že vezmem každý prvok daného poľa a priradím k nemu znak „!“. Nakoniec si vezmem vstupnú kombináciu „0“ a „1“ oddelených bodkami. Keď mám všetko rozdelené do polí (napísal som to presne v opačnom poradí ako to mám v kóde, ale na tom nezáleží) porovnávam jednotlivé polia. Vezmem prvé pole sčítancov a hľadám a porovnávam jednotlivé prvky v ňom s prvkami poľa pozitívnych a negatívnych písmen. Pre zjednodušenie uvediem príklad. Povedzme, že máme funkciu, ktorá rozdelila pole sčítancov na pole násobkov (neviem či som to správne nazval). Prvé pole násobkov môže vyzerať nejak takto „A!.B.C!“. Vezmem pole, v ktorom sú uložené hodnoty vstupných premenných. Následne pole „A!.B.C!“ rozdelím podľa znamienka „.“ a porovnávam jednotlivé jeho prvky s polom pre vstupné premenné. To robím tak, že vezmem pole s pozitívnymi a negatívnymi prvkami. Ak natrafím na prvok „A“ v danom reťazci, a hodnota pre „A“ je „0“, opúšťam toto pole násobkov a idem na ďalšie. Ak natrafím na prvok „A!“ a hodnota pre „A“ je „0“, zvyšujem pomocnú premennú. Ak táto premenná dosiahne hodnotu počtu prvkov v poli sčítancov, pole vyhodnotím ako kladné a teda celý výraz ako „1“. Takto prechádzam a overujem všetky možné kombínacie daných polí. Návratová hodnota tejto funkcie, slúžiacej na vyhodnotenie booleovského výrazu má návratovú hodnotu typu bool. Ak je návratová hodnota „true“, výraz sa vyhodnotil ako „1“, inak ako „0“.

* 1. **Generátor náhodných stringov - implementácia**

V projekte sa nachádza funkcia, ktorá generuje náhodné stringy, a ktorú som sám implementoval. Jej návratovou hodnotou je string vo formáte, s akým v celom projekte pracujem. Vstupnou hodnotou je integer, ktorý určuje počet písmen v stringu. Tie sa náhodne generujú naprieč celou abecedou a stáva sa, že sa nejaké z písmen opakuje. Taktiež sa náhodne generuje negácia a znamienka „.“ a „+“. Vzhľadom na to, že sa niekedy písmenká opakujú, nie vždy sa jedná o rovnaký počet vsutpných premenných. Napríklad pre výraz dĺžky 20 môžeme dostať 20 rozličných písmen, a teda 20 vstupných premenných, ale taktiež aj 10. Väčšinou sa to nejak spriemeruje a nejde to do extrémnych hodnôt. Takto vytvorené booleovské výrazy som si dal vypísať a vybral som prvých 100 s počtom premenných 13. Súčasťou generátora je aj funkcia, ktorej výslednou hodnotou je string s poradím premenných, v akom do výrazu vstupujú. Žiaľ, toto je vždy zoradené podľa abecedy, avšak nemalo by to byť prekážkou. Toto poradie je vo forme, s akou pracujem naprieč mojim programom. Následne generátor obsahuje textový súbor, ktorý som prebral od spolužiaka **Filip Zatroch**, v ktorom sa nachádzajú všetky možné kombinácie „0“ a „1“ pre 13 vstupných premenných oddelených znakom „.“, tzn. vo formáte v akom s nimi v programe pracujem.

* 1. **Tester a testovanie**

Neoddeliteľnou súčasťou môjho projektu je tester. Ten funguje na báze vloženia 100 booleovských výrazov, ktoré som si pomocou generátoru nechal vypísať a uložiť do poľa do funckii. Následne tester vygeneruje poradie týchto premenných podľa abecedy a spustí porovnávanie pre všetky možné kombinácie „0“ a „1“ na vstupe. Ak sa výsledok funkcie na vyhodnotenie booleovského výrazu nezhoduje s diagramom, zvýši pomocnú premennú „nezhoda“ o 1. S čistým svedomím môžem prehlásiť, že táto premenná zostala pre všetkých 100 trinásťvstupových premenných nulová pri každej vetve. To znamená, že testovanie prebehlo bez jedinej chyby. Celý tester som vytvoril spojením všetkych implementovaných funkcii, tzn. na vytvorenie diagramu, prehľadávanie diagramu, funkciu na vyhodnotenie zadaného booleovského výrazu, generátoru náhodných stringov vrátane všetkých pomocných funkcií, ktoré som v mojom programe použil a jeden main, ktorý zabezpečuje volanie všetkých týchto funkcii.

* 1. **Miera redukcie**

Súčasťou testera je aj výpis počtu uzlov, ktoré boli vytvorené, a ktoré som následne porovnal s celkovým počtom uzlov pre daný diagram. Nasledujúci graf zobrazuje mieru zredukovania pre 100 testovaných booleovských výrazov.

Graf zobrazuje mieru zredukovania v percentách. Jeden stĺpec predstavuje počet všetkých uzlov pre daný diagram, pričom oranžovou farbou je vyznačený počet uzlov v mojom – redukovanom diagrame a modrou farbou sú vyjadrené uzly, ktoré boli odstránené. Priemerná miera redukcie pre 100 trinásťvstupových booleovských výrazov je **78,86%**.

Graf zobrazuje počet uzlov môjho redukovaného diagramu pre jednotlivé booleovské výrazy. Opäť bolo porovnávaných tých istých 100 trinásťvstupových výrazov. Najväčší počet vytvorených uzlov je **7167** a najmenší **303**.

**Záver**

Zámerom tohto projektu bolo vytvoriť binárny rozhodovací diagram, s redukciou počas behu programu. Do diagramu boli dosadené rôzne booleovské funkcie, ktoré mal za úlohu vyhodnotiť. Následné hodnoty boli porovnávané s výslednými hodnotami funkcie na výpočet booleovských výrazov. Zadanie hodnotím ako celkom náročné aj vzhľadom na to, že som nedokázal nájsť na internete dostatok užitočných informácií z môjho uhľa pohľadu. Riešenie tohto projektu mi zabralo 5 celých dní, avšak dalo mi to veľa čo sa týka dátových štruktúr a algoritmov. Bonusom je, že som mohol pracovať v C, respektíve procedurálnom C++, kde som zdokonalil svoje znalosti. Z tohto hľadiska hodnotím zadanie pozitívne.