Špecifikácia a implementácia vyvážených stromov v Peanovej aritmetike

BAKALÁRSKA PRÁCA

Matej Vince

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
KATEDRA INFORMATIKY

Študijný odbor: 9.2.1 Informatika

Vedúci bakalárskej práce: Mgr. Ján Kľuka

BRATISLAVA 2007

Vyhlásenie Čestne vyhlasujem, že bakalársku prácu som vypracoval samostatne len s použitím uvedenej literatúry.

V Bratislave dňa 14.6.2007

....

Poďakovanie Ďakujem svojmu vedúcemu Mgr. Jánovi Kľukovi za cenné rady, trpezlivosť a ochotu, ktoré mi pomohli pri písaní tejto práce.

Abstrakt

Vince, Matej. *Špecifikácia a implementácia vyvážených stromov v Peanovej aritmetike* [bakalárska práca]. Univerzita Komenského v Bratislave. Fakulta matematiky, fyziky a informatiky. Katedra informatiky. Školiteľ: Mgr. Ján Kľuka.

Táto práca sa zaoberá 2-3 stromami a červeno-čiernymi stromami. Implementuje tieto dátové štruktúry a rovnako aj základné operácie používané na prácu so stromami: príslušnosť k stromu, vkladanie prvku a odoberanie prvku. Obsahuje aj formálnu špecifikáciu týchto operácií, môže mať teda východiskovú pozíciu pri verifikácii ich implementácie. Práca tiež môže slúžiť na rozšírenie výučby deklaratívneho programovania.

This bachelor work studies 2-3 and red-black tree data structures. It implements these data structures and also basic operations with them: membership, insert and delete. Formal specification of these operations is included, so it can be used as a starting position for verification of its implementation. This work can be also used as an additional text to teaching declarative programming.

Obsah

1 Úvod

1.1 Cieľ

Cieľom tejto bakalárskej práce je implementovať niektoré vyvážené stromy v Peanovej aritmetike. Konkrétne sa jedná o červeno-čierne stromy a 2-3 stromy. Na stromy sa pozeráme ako na štruktúry reprezentujúce množiny prvkov. Implementujeme štandardné operácie, ktoré sa používajú na prácu so stromami. Operácia byť v strome ($x \in t$) znamená, zistiť či strom t obsahuje hodnotu x. Operácia vkladania prvku do stromu ($t \cup \{x\}$) vloží prvok x do stromu t, pričom zachová štruktúru stromu t. Nakoniec operácia odoberania prvku zo stromu ($t \setminus \{x\}$) odberie prvok x z t a tiež zachová štruktúru stromu. Tiež požadujeme, aby implementácia týchto operácií bola efektívna, čiže aby tieto operácie bežali v čase $O(\log n)$. Tieto operácie chceme špecifikovať v zmysle popísať ich vlastnosťami.

1.2 Motivácia

Existuje veľa učebníc, ktoré popisujú uvedené dátové štruktúry a operácie na nich, avšak popisujú ich v imperatívnych jazykoch, napríklad [1],[2]. Takisto nie sú dostupné alebo nie sú známe učebnice zaoberajúce sa deklaratívnym programovaním, ktoré by sa zaoberali týmito dátovými štruktúrami. Jedinou výnimkou je [3], tu sa však rieši táto problematika len okrajovo. Je tu opísané vkladanie do červeno-čierneho stromu, to je ale jednoduchšia z požadovaných operácií. Vznikla tak potreba ich implementácie v nejakom deklaratívnom jazyku. Ako implementačný jazyk použijeme jazyk CL, ktorý sa používa na našej fakulte na výučbu deklaratívneho programovania. Tento jazyk nám tiež umožňuje špecifikovať vlastnosti a verifikovať implementáciu vzhľadom na tieto vlastnosti. A ďalšou motiváciou je pripraviť dobré východisko na verifikovanie nami implementovaných algoritmov, vďaka tejto špecifikácii.

1.3 Metóda riešenia

Implementáciu samotných stromov riešime tak, že k obom typom si najprv zadefinujeme všeobecnejší model stromov: listové stromy a farbené stromy. Z listových stromov odvodíme 2-3 stromy pridaním požiadaviek na hĺbku listov, počet synov uzla a na usporiadanie hodnôt. Z farbených, pridaním podmienok na hĺbku, usporiadanie hodnôt a následnosť farieb zas červeno-čierne stromy. Pri implementácii vkladania a odoberania prvkov vychádzame z algoritmov pre imperatívne jazyky. Výstupom našej práce sú dva dokumenty. Tento text, ktorý slúži tiež ako rozšírenie [4]. A tiež zdrojový kód algoritmov v jazyku CL sformátovaný do XML, ktorý bude uverejnený na internete na adrese http://ii.fmph.uniba.sk/cl.

1.4 Štuktúra práce

V druhej kapitole sú zadefinované pomocné funkcie , ktoré budeme potrebovať pri implementácii stromov.V tretej kapitole sa zaoberáme listovými stromami a 2-3 stromami. V kapitole štyri sú farbené a červeno-čierne stromy.

2 Pomocné funkcie

V ďalšom texte budeme pri definíciách a popisovaní vlastností vychádzať z [4] a predpokladať, že čitateľ bol oboznámený so základnými pojmami z Peanovej aritmetiky a jazyka CL. Napriek tomu si však zadefinujeme niekoľko pomocných funkcií, ktoré neskôr využijeme.
Dĺžka zoznamu:

$$\begin{aligned}&\mathrm{L}(0) = 0\\&\mathrm{L}(u,v) = 1 + \mathrm{L}(v).\end{aligned}$$

Posledný prvok zoznamu má nasledovnú vlastnosť:

$$x \neq 0 \rightarrow \exists y\, y \oplus (\mathrm{Last}(x), 0) = x. \tag{1}$$

Rekurzívna definícia potom vyzerá nasledovne:

$$\begin{aligned}&\mathrm{Last}(0) = 0\\&\mathrm{Last}(u,0) = u\\&\mathrm{Last}(u,v) = \mathrm{Last}(v) \leftarrow v \neq 0.\end{aligned}$$

O maxime z dvoch prvkov vieme povedať:

$$\max(x,y) \geq x \wedge \max(x,y) \geq y \tag{2}$$
$$\max(x,y) = x \vee \max(x,y) = y \tag{3}$$

a funkciu definujeme nasledovne:

$$\begin{aligned}&\max(x,y) = x \leftarrow x \leq y\\&\max(x,y) = y \leftarrow x > y.\end{aligned}$$

Budeme tiež potrebovať i-ty prvok zoznamu, o ktorom má platiť:

$$\exists u \exists v (u \oplus (l_i, v) = l \wedge L(u) = i). \tag{4}$$

Rekurzívne ho definujeme:

$$\begin{aligned}&(u,v)_0 = u\\&(u,v)_{i+1} = v_i.\end{aligned}$$

3 2-3 stromy

3.1 Listové stromy (Leaf trees)

3.1.1 Konštruktory a formát. Pomocou B^+ stromov, zadefinovaných v [5], môžeme zadefinovať *listové stromy*. Rovnako ako B^+ stromy majú dáta uložené len v listoch. Líšia sa však v tom, že *listové stromy* nemajú tieto dáta usporiadané a tiež nemajú žiadne obmedzenia na hĺbku listov a počet synov. V našej definícii sme tiež vynechali kľúče, ktoré sú v B^+ stromoch uložené vo vnútorných uzloch.

Listové stromy aritmetizujeme do prirodzenych čísel troma konštruktormi:

$$\bullet = 0,0$$
$$[a] = 1,a$$
$$\frac{\wedge}{ts} = 2,ts.$$

Pričom prázdny strom aritmetizujeme konštruktorom $\bullet$, list konštruktorom $[a]$ a vnútorný uzol konštruktorom $\frac{\wedge}{ts}$, pričom ts je zoznam podstromov.

Zadefinujeme predikát $\mathrm{Leaftrees}$, ktorý je platný pre kódy zoznamov neprázdnych listových stromov:

$$\mathrm{Leaftrees}(0)$$
$$\mathrm{Leaftrees}([a],ts) \leftarrow \mathrm{N}(a) \wedge \mathrm{Leaftrees}(ts)$$
$$\mathrm{Leaftrees}(\frac{\wedge}{ts_1},ts) \leftarrow \mathrm{Leaftrees}(ts_1) \wedge \mathrm{Leaftrees}(ts)$$

a pomocou neho môžeme definovať samotný predikát platný pre listové stromy:

$$\mathrm{Leaftree}(t) \leftrightarrow t = \bullet \vee \mathrm{Leaftrees}(t,0).$$

Stotožníme listové stromy s ich kódmi a odteraz budeme vravieť *listový strom t* namiesto *kód listového stromu t*.

3.1.2 Pomocné funkcie a predikáty. Zadefinujeme niekoľko štandardných funkcií a predikátov pre listové stromy.
Predikát $\mathrm{Is\_leaf}\,(x)$ je platný, ak x je listom, čo sa dá jednoducho zistiť:

$$\mathrm{Is\_leaf}\,(x) \leftarrow x = [y].$$

Príslušnosť k listovému stromu označíme $\in_{\mathrm{lt}}$.Predikát $x \in_{\mathrm{lt}} t$ platí ak strom t obsahuje hodnotu x. Rekurzívne ho definujeme nasledovne:

$$x \in_{\mathrm{lt}} [a] \leftarrow a = x$$
$$x \in_{\mathrm{lt}} \frac{\wedge}{t_1, ts_1} \leftarrow x \in_{\mathrm{lt}} t_1$$
$$x \in_{\mathrm{lt}} \frac{\wedge}{t_1, ts_1} \leftarrow \neg x \in_{\mathrm{lt}} t_1 \wedge x \in_{\mathrm{lt}} \frac{\wedge}{ts_1}.$$

Keďže listové stromy nemajú pevne stanovený počet synov, zadefinujeme funkciu $\mathrm{Degree}(t)$, ktorá nám vráti počet synov stromu t:

$$\mathrm{Degree}(\bullet) = 0$$
$$\mathrm{Degree}\,[x] = 0$$
$$\mathrm{Degree}\,\frac{\wedge}{ts} = \mathrm{L}(ts).$$

Funkcia $\mathrm{Max}_{\mathrm{lt}}\,(t)$ vráti najväčšiu hodnotu stromu t alebo 0 ak je to prázdny strom. Požadujeme:

$$\mathrm{Leaftree}(t) \rightarrow \mathrm{Max}_{\mathrm{lt}}\,(t) \in_{\mathrm{lt}} t \tag{1}$$
$$\mathrm{Leaftree}(t) \wedge x \in_{\mathrm{lt}} t \rightarrow x \leq \mathrm{Max}_{\mathrm{lt}}\,(t). \tag{2}$$

V rekurzívnej definícii prechádzame všetky podstromy a vrátime maximum z nich:

$$\mathrm{Max}_{\mathrm{lt}}\,(\bullet) = 0$$
$$\mathrm{Max}_{\mathrm{lt}}\,[x] = x$$
$$\mathrm{Max}_{\mathrm{lt}}\,\frac{\wedge}{0} = 0$$
$$\mathrm{Max}_{\mathrm{lt}}\,\frac{\wedge}{t_1, 0} = \mathrm{Max}_{\mathrm{lt}}\,(t_1)$$
$$\mathrm{Max}_{\mathrm{lt}}\,\frac{\wedge}{t_1, t_2, ts} = \max(\mathrm{Max}_{\mathrm{lt}}\,(t_1), \mathrm{Max}_{\mathrm{lt}}\,\frac{\wedge}{t_2, ts}).$$

Za zmienku stojí posledná klauzula. Keďže CL neumožňuje vzájomnú rekurziu, nemôžeme mať funkciu ktorá by hľadala maximum zo zoznamu stromov, lebo by musela volať Maxlt_{lt}. Namiesto toho zisťujeme maximum prvého podstromu a porovnávame ho s maximom zvyšných podstromov. Podobnú konštrukciu použijeme aj na ďalších miestach.
Hĺbka stromu $\mathrm{Depth}(t)$ je najdlhšia cesta v strome t, spomedzi všetkých ciest vedúcich z koreňa do listu:

$$\mathrm{Depth}(\bullet) = 0$$

$$
\begin{aligned}
&\mathrm{Depth}\,[a] = 1\\
&\mathrm{Depth}\,\frac{\wedge}{t_1, 0} = 1 + \mathrm{Depth}(t_1)\\
&\mathrm{Depth}\,\frac{\wedge}{t_1, t_2, ts} = \max(1 + \mathrm{Depth}(t_1), \mathrm{Depth}\,\frac{\wedge}{t_2, ts}).
\end{aligned}
$$

Počet vrcholov stromu $\mathrm{Sz}(t)$ je počet vnútorných uzlov a listov:

$$
\begin{aligned}
&\mathrm{Sz}(\bullet) = 0\\
&\mathrm{Sz}\,[a] = 1\\
&\mathrm{Sz}\,\frac{\wedge}{0} = 1\\
&\mathrm{Sz}\,\frac{\wedge}{t_1, ts} = \mathrm{Sz}(t_1) + \mathrm{Sz}\,\frac{\wedge}{ts}.
\end{aligned}
$$

Predikát $t \preceq m$ platí, ak všetky prvky v strome t sú menšie alebo rovné ako m, inak povedané:

$$
\mathrm{Leaftree}(t) \rightarrow t \preceq m \leftrightarrow \forall x (x \in_{\mathrm{lt}} t \rightarrow x \leq m), \tag{3}
$$

a rekurzívne zapísané:

$$
\begin{aligned}
&[a] \preceq m \leftarrow a \leq m\\
&\frac{\wedge}{0} \preceq m\\
&\frac{\wedge}{t_1, ts} \preceq m \leftarrow t_1 \preceq m \wedge \frac{\wedge}{ts} \preceq m.
\end{aligned}
$$

Podobne predikát $t \succ m$ platí, ak sú všetky prvky v t väčšie ako m:

$$
\mathrm{Leaftree}(t) \rightarrow t \succ m \leftrightarrow \forall x (x \in_{\mathrm{lt}} t \rightarrow x > m). \tag{4}
$$

Rekurzívna defínícia je analogická:

$$
\begin{aligned}
&[a] \succ m \leftarrow a > m\\
&\frac{\wedge}{0} \succ m\\
&\frac{\wedge}{t_1, ts} \succ m \leftarrow t_1 \succ m \wedge \frac{\wedge}{ts} \succ m.
\end{aligned}
$$

Predikát $st \trianglelefteq t$ platí, ak st je podstromom stromu t:

$$
\begin{aligned}
&\mathrm{Leaftree}(st) \wedge \mathrm{Leaftree}(t) \rightarrow\\
&st \trianglelefteq t \leftrightarrow st = t \vee \exists t_1 \exists st \big(t = \frac{\wedge}{ts} \wedge t_1 \,\varepsilon\, ts \wedge st \trianglelefteq t_1\big).
\end{aligned} \tag{5}
$$

Rekurzívne ho zadefinujeme takto:

$$
\begin{aligned}
&st \trianglelefteq t \;\leftarrow\; t = st\\
&st \trianglelefteq t \;\leftarrow\; t \neq st \wedge t = \frac{\wedge}{t_1, ts} \wedge st \trianglelefteq t_1\\
&st \trianglelefteq t \;\leftarrow\; t \neq st \wedge t = \frac{\wedge}{t_1, ts} \wedge \neg st \trianglelefteq t_1 \wedge st \trianglelefteq \frac{\wedge}{ts}.
\end{aligned}
$$

Treba si uvedomiť, že žiadny neprázdnu neobsahuje $\bullet$ a teda $\bullet$ je podstromom len samého seba.

3.2 2-3 stromy

V nasledujúcom texte až do konca kapitoly budeme vychádzať z [2].

3.2.1 Definícia. 2-3 stromy sú listové stromy, ktoré spĺňajú nasledovné požiadavky:

1. každý vnútorný uzol okrem listov má dvoch alebo troch synov,
2. všetky listy majú rovnakú hĺbku,
3. hodnoty v strome sú usporiadané zľava doprava.

Na rozdiel od implementácie v imperatívnych jazykoch, sme tu však vynechali vlastnosť, že vnútorné uzly obsahujú uložené maximá prvého a druhého syna. Pôvodne naša definícia obsahovala aj tieto maximá, postupom času sme ich však vynechali. Týmto krokom sme síce zhoršili celkovú zložitosť vkladania a odoberania prvkov z $O(\log n)$ na $O(\log^2 n)$, na druhej strane sme ale výrazne zlepšili čitateľnosť samotných algoritmov, čo má v našom prípade väčší význam. Vyváženosť takéhoto stromu nám zabezpečuje druhá vlastnosť.

3.2.2 Formalizácia definície. Najskôr zadefinujeme predikát $\mathrm{Stree}_{23}(t)$, platný pre neprázdny 2-3 strom t:

$$
\begin{aligned}
&\mathrm{Stree}_{23}\,[a]\\
&\mathrm{Stree}_{23}\,\frac{\wedge}{t_1, t_2, 0} \leftarrow\\
&\qquad \mathrm{Max}_{\mathrm{lt}}(t_1) < \mathrm{Max}_{\mathrm{lt}}(t_2) \wedge t_1 \preceq \mathrm{Max}_{\mathrm{lt}}(t_1) \wedge t_2 \succ \mathrm{Max}_{\mathrm{lt}}(t_1) \wedge\\
&\qquad t_2 \preceq \mathrm{Max}_{\mathrm{lt}}(t_2) \wedge \mathrm{Stree}_{23}(t_1) \wedge \mathrm{Stree}_{23}(t_2) \wedge \mathrm{Depth}(t_1) = \mathrm{Depth}(t_2)
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{Stree}_{23}\,\frac{\wedge}{t_1,t_2,t_3,0} \leftarrow \\
&\quad \mathrm{Max}_{\mathrm{lt}}(t_1) < \mathrm{Max}_{\mathrm{lt}}(t_2) \wedge \mathrm{Max}_{\mathrm{lt}}(t_2) < \mathrm{Max}_{\mathrm{lt}}(t_3) \wedge t_1 \preceq \mathrm{Max}_{\mathrm{lt}}(t_1) \wedge \\
&\quad t_2 \succ \mathrm{Max}_{\mathrm{lt}}(t_1) \wedge t_2 \preceq \mathrm{Max}_{\mathrm{lt}}(t_2) \wedge t_3 \succ \mathrm{Max}_{\mathrm{lt}}(t_2) \wedge \\
&\quad t_3 \preceq \mathrm{Max}_{\mathrm{lt}}(t_3) \wedge \mathrm{Stree}_{23}(t_1) \wedge \mathrm{Stree}_{23}(t_2) \wedge \mathrm{Stree}_{23}(t_3) \wedge \\
&\quad \mathrm{Depth}(t_1) = \mathrm{Depth}(t_2) \wedge \mathrm{Depth}(t_2) = \mathrm{Depth}(t_3)
\end{aligned}
$$

a pomocou neho môžeme zadefinovať predikát $\mathrm{Tree}_{23}(t)$, ktorý platí pre ľubovoľný 2-3 strom:

$$
\begin{aligned}
&\mathrm{Tree}_{23}(t) \leftarrow t = \bullet \\
&\mathrm{Tree}_{23}(t) \leftarrow t \neq \bullet \wedge \mathrm{Stree}_{23}(t).
\end{aligned}
$$

$\mathrm{Tree}_{23}(t)$ spĺňa nasledovnú vlastnosť

$$\mathrm{Tree}_{23}(t) \rightarrow \mathrm{Leaftree}(t) \tag{1}$$

Ďalej potrebujeme predikát $\mathrm{Forest}_{23}(t)$, ktorý je platný pre zoznam 2-3 stromov a spĺňa nasledovnú vlastnosťou

$$\mathrm{Forest}_{23}(ts) \wedge (\mathrm{L}(ts) = 2 \vee \mathrm{L}(ts) = 3) \leftrightarrow \mathrm{Stree}_{23}\,\frac{\wedge}{ts}. \tag{2}$$

Rekurzívna definícia:

$$
\begin{aligned}
&\mathrm{Forest}_{23}(t_1,0) \leftarrow \mathrm{Stree}_{23}(t_1) \\
&\mathrm{Forest}_{23}(t_1,t_2,ts) \leftarrow \mathrm{Stree}_{23}(t_1) \wedge \mathrm{Stree}_{23}(t_2) \wedge \mathrm{Max}_{\mathrm{lt}}(t_1) < \mathrm{Max}_{\mathrm{lt}}(t_2) \wedge \\
&\qquad\qquad \mathrm{Depth}(t_1) = \mathrm{Depth}(t_2) \wedge \mathrm{Forest}_{23}(t_2,ts).
\end{aligned}
$$

Špeciálny prípad $\mathrm{Forest}_{23}(t)$ je $\mathrm{Leaves}_{23}(t)$, kde každý prvok zoznamu bude list:

$$\mathrm{Leaves}_{23}(ls) \leftrightarrow \mathrm{Forest}_{23}(ls) \wedge \forall l(l \,\varepsilon\, ls \rightarrow \mathrm{Is\_leaf}(l))$$

Z uvedeného dostaneme vlastnosť:

$$\mathrm{Stree}_{23}\,\frac{\wedge}{t,ts} \wedge \mathrm{Is\_leaf}(t) \rightarrow \mathrm{Leaves}_{23}(t,ts) \tag{3}$$

Tým, že 2-3 stromy majú obmedzený počet synov uzla, môžeme jednoducho vyjadriť vzťah medzi počtom vrcholov a hĺbkou stromu:

$$\mathrm{Tree}_{23}(t) \rightarrow 2^{\mathrm{Depth}(t)} \dot{-} 1 \leq \mathrm{Sz}(t) \wedge \mathrm{Sz}(t) \leq (3^{\mathrm{Depth}(t)} \dot{-} 1) \div 2 \tag{4}$$

3.2.3 Prehľadávanie. Skôr ako budeme testovať samotnú príslušnosť prvku k 2-3 stromu, zadefinujeme maximum stromu, pomocou ktorého ju môžeme testovať efektívnejšie. Maximum môžeme hľadať efektívnejšie vďaka tomu, že hodnoty v strome sú usporiadané. Musí ale zostať platné:

$$\mathrm{Tree}_{23}(t) \;\rightarrow\; \mathrm{Max}_{23}(t) = x \leftrightarrow \mathrm{Max}_{\mathrm{lt}}(t) = x. \tag{1}$$

Vďaka tomu, že podstromy sú usporiadané, môžeme hľadať maximum rovno v poslednom synovi

$$\begin{aligned}
&\mathrm{Max}_{23}(\bullet) = 0\\
&\mathrm{Max}_{23}\,[x] = x\\
&\mathrm{Max}_{23}\,\frac{\wedge}{ts} = \mathrm{Max}_{23}\,\mathrm{Last}(ts).
\end{aligned}$$

Teraz už môžeme efektívne overiť príslošnosť prvku k stromu. Obdobne tu platí vlastnosť:

$$\mathrm{Tree}_{23}(t) \;\rightarrow\; x \in_{23} t \leftrightarrow x \in_{\mathrm{lt}} t \tag{2}$$

Samotnú príslušnosť overíme spôsobom podobným binárnemu prehľadávaniu. Strom t prehľadáme od koreňa. Najskôr hľadaný prvok porovnáme s maximom prvého podstromu. Ak je menší, tak testujeme príslušnosť k tomuto podstromu. Ak je rovný maximu, znamená to, že v podstrome sa určite nachádza. Ak je väčší, porovnáme ho s maximom ďalšieho podstromu:

$$\begin{aligned}
&x \in_{23} [a] \leftarrow a = x\\
&x \in_{23} \frac{\wedge}{t,ts} \leftarrow x < \mathrm{Max}_{23}(t) \wedge x \in_{23} t\\
&x \in_{23} \frac{\wedge}{t,ts} \leftarrow x = \mathrm{Max}_{23}(t)\\
&x \in_{23} \frac{\wedge}{t,ts} \leftarrow x > \mathrm{Max}_{23}(t) \wedge x \in_{23} \frac{\wedge}{ts}.
\end{aligned}$$

Predikát $\mathrm{In\_forest}_{23}(x,ts)$ je platný, ak sa hodnota x nachádza v niektorom strome zo zoznamu ts. Postupne prechádzame zoznam a testovať príslušnosť nám stačí v prvom strome takom, že jeho maximum je väčšie ako x:

$$\begin{aligned}
&\mathrm{In_forest}_{23}(x,t,ts) \leftarrow x \leq \mathrm{Max}_{23}(t) \wedge x \in_{23} t\\
&\mathrm{In_forest}_{23}(x,t,ts) \leftarrow x > \mathrm{Max}_{23}(t) \wedge \mathrm{In_forest}_{23}(x,ts).
\end{aligned}$$

Pre tento predikát platí vlastnosť:

$$\mathrm{Forest}_{23}(ts) \;\rightarrow\; \mathrm{In\_forest}_{23}(x,ts) \leftrightarrow x \in_{\mathrm{lt}} \frac{\wedge}{ts}. \tag{3}$$

3.2.4 Insert. Ďalej môžeme pristúpiť k vkladaniu prvku do 2-3 stromu. Pri vkladaní prvku najprv nájdeme miesto kde by sa mal prvok nachádzať a na tomto mieste pridáme list s danou hodnotou. Môže sa nám ale stať, že porušíme štruktúru 2-3 stromu. Ak mal uzol, pod ktorý vkladáme nový list, troch synov, potom dostávame uzol zo štyrmi synmi. Keďže nemáme k dispozícii smerníky a teda ani smerník priamo na otca, nevieme túto situáciu riešiť hneď, budeme ju riešiť až pri vynáraní sa z rekurzie.

Najprv musíme zistiť, či daný uzol porušuje podmienku o počte synov. Na to použijeme nasledovný predikát:

$$\mathrm{T_{23}i}\,\frac{\wedge}{ts} \leftarrow \mathrm{Forest_{23}}\,(ts) \wedge \mathrm{Degree}\,\frac{\wedge}{ts} = 4.$$

Samotné fixovanie uzla nie je zložité. Musí spĺňať nasledovné vlastnosti:

$$\mathrm{Tree_{23}}\,(t) \;\rightarrow\; \mathrm{Insfix}(t) = t, 0 \tag{1}$$
$$\mathrm{T_{23}i}\,(t) \;\rightarrow\; \mathrm{Forest_{23}}\,\mathrm{Insfix}(t) \wedge \mathrm{L}\,\mathrm{Insfix}(t) = 2 \tag{2}$$
$$\mathrm{T_{23}i}\,(t) \;\rightarrow\; x \in_{23} t \leftrightarrow \mathrm{In\_forest_{23}}\,(x, \mathrm{Insfix}(t)) \tag{3}$$
$$\mathrm{T_{23}i}\,(t) \wedge t_1 \;\varepsilon\; \mathrm{Insfix}(t) \;\rightarrow\; \mathrm{Depth}(t_1) = \mathrm{Depth}(t). \tag{4}$$

Pri opravovaní štruktúry teda môžu nastať dve možnosti. Buď daný uzol je platným podstromom nejakého 2-3 stromu a vtedy uzol nemusíme nijako fixovať, alebo je to uzol zo štyrmi synmi. Túto situáciu riešime tak, že daný uzol rozdelíme na dva nové uzly, pričom prvý obsahuje prvých dvoch a druhý posledných dvoch synov pôvodného uzla. Takto zachováme hĺbky a aj všetky prvky. Keďže nevieme, či nám daná funkcia vráti jeden alebo dva prvky a aby sme to potom nemuseli rozoberať na jednotlivé prípady, vrátime zoznam prvkov:

$$\mathrm{Insfix}(t) = t, 0 \leftarrow \mathrm{Degree}(t) < 4$$
$$\mathrm{Insfix}(t) = \frac{\wedge}{t_1, t_2, 0}, \frac{\wedge}{t_3, t_4, 0}, 0 \leftarrow \mathrm{Degree}(t) = 4 \wedge t = \frac{\wedge}{t_1, t_2, t_3, t_4, 0}$$

Samotné pridávanie prvku bude prebiehať na najnižšej úrovni, budeme ho pridávať do zoznamu listov. Funkcia musí spĺňať:

$$\mathrm{Leaves_{23}}\,(t) \;\rightarrow\; \mathrm{Leaves_{23}}\,\mathrm{Insleaf}(t, x) \tag{5}$$
$$\mathrm{Leaves_{23}}\,(t) \;\rightarrow\; \mathrm{In\_forest_{23}}\,(y, \mathrm{Insleaf}(t, x)) \leftrightarrow \mathrm{In\_forest_{23}}\,(y, t) \vee y = x. \tag{6}$$

Postupne prechádzame zoznam a pridáme prvok na správne miesto, samozrejme, ak sa tam nachádza, nepridáme nič:

$$\mathrm{Insleaf}(0, x) = [x], 0$$

$$\mathrm{Insleaf}(([a],ls),x) = [a], \mathrm{Insleaf}(ls,x) \leftarrow a < x$$
$$\mathrm{Insleaf}(([a],ls),x) = [a], ls \leftarrow a = x$$
$$\mathrm{Insleaf}(([a],ls),x) = [x], [a], ls \leftarrow a > x.$$

Teraz potrebujeme vložiť prvok do neprázdneho podstromu, pričom chceme, aby výsledkom bol buď dobrý podstrom, ale nie list, alebo strom, ktorý má štyroch synov. Tiež chceme, aby sa nestratili ani nepribudli žiadne prvky a aby sa zachovali hĺbky. Formálne popísané:

$$\begin{aligned}\mathrm{Stree}_{23}\,(t) \;\rightarrow &\mathrm{Stree}_{23}\,\mathrm{Insert}_1\,(t,x) \wedge \neg\mathrm{Is\_leaf}\,\mathrm{Insert}_1\,(t,x) \vee \\ &\mathrm{T}_{23}\mathrm{i}\,\mathrm{Insert}_1\,(t,x)\end{aligned} \tag{7}$$

$$\mathrm{Stree}_{23}\,(t) \;\rightarrow\; y \in_{23} \mathrm{Insert}_1\,(t,x) \leftrightarrow y \in_{23} t \vee x = y \tag{8}$$

$$\mathrm{Stree}_{23}\,(t) \wedge \neg\mathrm{Is\_leaf}\,(t) \;\rightarrow\; \mathrm{Depth}(t) = \mathrm{Depth}\,\mathrm{Insert}_1\,(t,x) \tag{9}$$

Pridávať budeme tiež spôsom ako pri prehľadávacích stromoch. Vkladanie do listu je zrejmé. Pri vkladaní do uzla najprv zisíme či uzol obsahuje listy, ak áno, pomocou predošlej funkcie vložíme prvok. Ak nie, prvok vložíme do prvého podstromu, ktorý má väčšie maximum. Keďže toto nám môže vrátiť uzol zo štyrmi synmi musíme zavolať funkciu Insfix na tomto podstrome. Keďže od nej sme požadovali zoznam, tak pridávanie výsledku medzi ostatných synov je jednoduché a robíme ho cez štandardné operácie na zoznamoch. Týmto dostaneme výsledok, aký sme požadovali:

$$\mathrm{Insert}_1\,([a],x) = \frac{\wedge}{\mathrm{Insleaf}(([a],0),x)}$$
$$\mathrm{Insert}_1\left(\frac{\wedge}{t_1,ts},x\right) = \frac{\wedge}{\mathrm{Insleaf}((t_1,ts),x)} \leftarrow \mathrm{Is\_leaf}\,(t_1)$$
$$\mathrm{Insert}_1\left(\frac{\wedge}{t_1,ts},x\right) = \frac{\wedge}{\mathrm{Insfix}\,\mathrm{Insert}_1\,(t_1,x) \oplus ts} \leftarrow$$
$$\neg\mathrm{Is\_leaf}\,(t_1) \wedge x \leq \mathrm{Max}_{23}\,(t_1)$$
$$\mathrm{Insert}_1\left(\frac{\wedge}{t_1,ts},x\right) = \frac{\wedge}{t_1, \mathrm{Insfix}\,\mathrm{Insert}_1\,(t_2,x)} \leftarrow$$
$$\neg\mathrm{Is\_leaf}\,(t_1) \wedge x > \mathrm{Max}_{23}\,(t_1) \wedge ts = t_2, 0$$
$$\mathrm{Insert}_1\left(\frac{\wedge}{t_1,ts},x\right) = \frac{\wedge}{t_1, \mathrm{Insfix}\,\mathrm{Insert}_1\,(t_2,x) \oplus (t_3,0)} \leftarrow$$
$$\neg\mathrm{Is\_leaf}\,(t_1) \wedge x > \mathrm{Max}_{23}\,(t_1) \wedge ts = t_2, t_3, 0 \wedge x \leq \mathrm{Max}_{23}\,(t_2)$$
$$\mathrm{Insert}_1\left(\frac{\wedge}{t_1,ts},x\right) = \frac{\wedge}{t_1, t_2, \mathrm{Insfix}\,\mathrm{Insert}_1\,(t_3,x)} \leftarrow$$
$$\neg\mathrm{Is\_leaf}\,(t_1) \wedge x > \mathrm{Max}_{23}\,(t_1) \wedge ts = t_2, t_3, 0 \wedge x > \mathrm{Max}_{23}\,(t_2).$$

Nakoniec vkladanie do stromu musí spĺňať štandardné vlastnosti:

$$\mathrm{Tree}_{23}\,(t) \;\to\; \mathrm{Stree}_{23}\; t \cup \{x\} \tag{10}$$

$$\mathrm{Tree}_{23}\,(t) \;\to\; y \in_{23} t \cup \{x\} \leftrightarrow y \in_{23} t \vee x = y. \tag{11}$$

Teda pridanie do prázdneho stromu je zrejmé a pridanie do neprázdneho vyrieši $\mathrm{Insert}_1\,(t,x)$. Ešte zostáva vyriešiť prípad, ak $\mathrm{Insert}_1\,(t,x)$ vrátil strom, ktorého koreň má štyroch synov. To vyriešime rovnako ako v $\mathrm{Insert}_1\,(t,x)$ zavolaním $\mathrm{Insfix}(t)$. To je vlastne jediný prípad, kde sa môže meniť hĺbka stromu (okrem pridávania do listu):

$$
\begin{aligned}
&t \cup \{x\} = [x] \;\leftarrow\; t = \bullet\\
&t \cup \{x\} = \mathrm{Insert}_1\,(t,x) \;\leftarrow\; t \neq \bullet \wedge \mathrm{Insert}_1\,(t,x) = z \wedge \mathrm{Degree}(z) < 4\\
&t \cup \{x\} = \frac{\wedge}{q} \;\leftarrow\; t \neq \bullet \wedge \mathrm{Insert}_1\,(t,x) = z \wedge \mathrm{Degree}(z) = 4 \wedge \mathrm{Insfix}(z) = q
\end{aligned}
$$

Ešte pre jednoduché pridávanie do stromu zadefinujeme funkciu, ktorá dostane na vstup strom a zoznam prvkov a všetky prvky zo zoznamu pridá do stromu. Chceme teda:

$$\mathrm{Tree}_{23}\,(t) \;\to\; \mathrm{Stree}_{23}\; \mathrm{Insert\_list}(t,ls) \tag{12}$$

$$\mathrm{Tree}_{23}\,(t) \;\to\; x \in_{23} \mathrm{Insert\_list}(t,ls) \leftrightarrow x \in_{23} t \vee x \,\varepsilon\, ls. \tag{13}$$

Prechádzame zoznam a prvky budeme pridávať po jednom:

$$
\begin{aligned}
&\mathrm{Insert\_list}(t,0) = t\\
&\mathrm{Insert\_list}(t,l,ls) = \mathrm{Insert\_list}(t \cup \{l\},ls).
\end{aligned}
$$

3.2.5 Delete. Odoberanie prvku z 2-3 stromu sa robí podobným princípom ako vkladanie. Tiež pri odoberaní sa nám môže stať, že dočasne porušíme štruktúru 2-3 stromu. Ak odoberieme podstrom z uzla, ktorý mal dvoch synov, dostávame uzol s jediným synom.

Potrebujeme predikát na zistenie, či daný uzol porušuje štruktúru:

$$\mathrm{T}_{23}\mathrm{d}\,\frac{\wedge}{t,0} \;\leftarrow\; \mathrm{Stree}_{23}\,(t).$$

Pri opravovaní štruktúry, na rozdiel od opravy pri vkladaní, musíme rozobrať dva prípady, a to či uzol má ľavého resp. pravého brata. Ak má obidvoch, tak nie podstatné, ktorého si vyberieme.

Rozoberme podrobne prípad, ak ľavý z dvojice môže byť poškodený. Musí platiť:

$$\mathrm{Stree}_{23}\,(t_1) \wedge \mathrm{Stree}_{23}\,(t_2) \;\rightarrow\; \mathrm{Delfixl}\,(t_1,t_2) = t_1,t_2,0 \tag{1}$$

$$\begin{aligned} \mathrm{T}_{23}\mathrm{d}\,(t_1) \wedge \mathrm{Stree}_{23}\,(t_2) \wedge \mathrm{Depth}(t_1) = \mathrm{Depth}(t_2) \;\rightarrow\; \mathrm{Forest}_{23}\,\mathrm{Delfixl}\,(t_1,t_2) \wedge \\ (\mathrm{L}\,\mathrm{Delfixl}\,(t_1,t_2) = 2 \vee \mathrm{L}\,\mathrm{Delfixl}\,(t_1,t_2) = 3) \end{aligned} \tag{2}$$

$$\begin{aligned} \mathrm{T}_{23}\mathrm{d}\,(t_1) \wedge \mathrm{Stree}_{23}\,(t_2) \wedge \mathrm{Depth}(t_1) = \mathrm{Depth}(t_2) \;\rightarrow \\ \mathrm{In\_forest}_{23}\,(x, \mathrm{Delfixl}\,(t_1,t_2)) \leftrightarrow \mathrm{In\_forest}_{23}\,(x,t_1,t_2,0) \end{aligned} \tag{3}$$

$$\begin{aligned} \mathrm{T}_{23}\mathrm{d}\,(t_1) \wedge \mathrm{Stree}_{23}\,(t_2) \wedge \mathrm{Depth}(t_1) = \mathrm{Depth}(t_2) \wedge t \;\varepsilon\; \mathrm{Delfixl}\,(t_1,t_2) \;\rightarrow \\ \mathrm{Depth}(t) = \mathrm{Depth}(t_1). \end{aligned} \tag{4}$$

Výsledkom funkcie bude opäť zoznam dĺžky jeden alebo dva. Ak ľavý nebol poškodený, tak vráti zoznam pôvodných stromov. Ak bol poškodený tak nastane jedna z dvoch situácií: pravý má dvoch alebo troch synov. Ak dvoch, tak spojíme oba uzly do jedného s tromi synmi. Ak má troch, tak odoberieme prvého z nich a pripojíme ho ako posledného syna poškodeného podstromu. Zjavne nezmeníme výšku a prvky sa zachovajú:

$$\begin{aligned} &\mathrm{Delfixl}\left(\frac{\wedge}{ts_1},\frac{\wedge}{ts_2}\right) = \frac{\wedge}{ts_1},\frac{\wedge}{ts_2},0 \leftarrow \mathrm{L}(ts_1) > 1 \\ &\mathrm{Delfixl}\left(\frac{\wedge}{ts_1},\frac{\wedge}{ts_2}\right) = \frac{\wedge}{ts_1 \oplus ts_2},0 \leftarrow \mathrm{L}(ts_1) = 1 \wedge \mathrm{L}(ts_2) = 2 \\ &\mathrm{Delfixl}\left(\frac{\wedge}{ts_1},\frac{\wedge}{ts_2}\right) = \frac{\wedge}{ts_1 \oplus (t_1,0)},\frac{\wedge}{t_2,t_3,0},0 \leftarrow \mathrm{L}(ts_1) = 1 \wedge \mathrm{L}(ts_2) = 3 \wedge \\ &\qquad\qquad ts_2 = t_1,t_2,t_3,0. \end{aligned}$$

Obdobne potom vyriešime, ak pravý syn je poškodený.

$$\begin{aligned} &\mathrm{Delfixr}\left(\frac{\wedge}{ts_1},\frac{\wedge}{ts_2}\right) = \frac{\wedge}{ts_1},\frac{\wedge}{ts_2},0 \leftarrow \mathrm{L}(ts_2) > 1 \\ &\mathrm{Delfixr}\left(\frac{\wedge}{ts_1},\frac{\wedge}{ts_2}\right) = \frac{\wedge}{ts_1 \oplus ts_2},0 \leftarrow \mathrm{L}(ts_2) = 1 \wedge \mathrm{L}(ts_1) = 2 \\ &\mathrm{Delfixr}\left(\frac{\wedge}{ts_1},\frac{\wedge}{ts_2}\right) = \frac{\wedge}{t_1,t_2,0},\frac{\wedge}{t_3,ts_2},0 \leftarrow \mathrm{L}(ts_2) = 1 \wedge \mathrm{L}(ts_1) = 3 \wedge \\ &\qquad\qquad ts_1 = t_1,t_2,t_3,0. \end{aligned}$$

Samotné odoberanie riešime na najnižšej úrovni. Ak sa tu daný prvok nenachádza, tak nič nezmeníme:

$$\begin{aligned} \mathrm{Leaves}_{23}\,(t) \;\rightarrow\; \mathrm{Leaves}_{23}\,\mathrm{Delleaf}(t,x) \wedge \\ (\mathrm{L}\,\mathrm{Delleaf}(t,x) = \mathrm{L}(t) \vee \mathrm{L}\,\mathrm{Delleaf}(t,x) + 1 = \mathrm{L}(t)) \end{aligned} \tag{5}$$

$$\mathrm{Leaves}_{23}(t) \;\rightarrow\; \mathrm{In\_forest}_{23}(y, \mathrm{Delleaf}(t,x)) \leftrightarrow \mathrm{In\_forest}_{23}(y,t) \wedge y \neq x. \tag{6}$$

Prechádzame listy pokiaľ neobsahujú väčšiu hodnotu ako tú, ktorú chceme odstrániť. Keď ju nájdeme, odstránime príslušný list:

$$\begin{aligned}
&\mathrm{Delleaf}(0,x) = 0\\
&\mathrm{Delleaf}(([a],ts),x) = [a],ts \;\leftarrow\; x < a\\
&\mathrm{Delleaf}(([a],ts),x) = ts \;\leftarrow\; x = a\\
&\mathrm{Delleaf}(([a],ts),x) = [a],\mathrm{Delleaf}(ts,x) \;\leftarrow\; x > a.
\end{aligned}$$

Teraz chceme odobrať prvok z podstromu, ktorý nie je list. Takisto ako pri vkladaní, výstup má byť podstrom, ktorý nie je list alebo podstrom s jedným synom, pričom zachováme hĺbky a všetky prvky okrem odstraňovaného. Presnejšie zapísané:

$$\begin{aligned}
\mathrm{Stree}_{23}(t) \wedge \neg\mathrm{Is_leaf}(t) \;\rightarrow\; & \mathrm{Stree}_{23}\,\mathrm{Delete}_1(t,x) \wedge \neg\mathrm{Is_leaf}(t) \vee \\
& \mathrm{T}_{23}\mathrm{d}\,\mathrm{Delete}_1(t,x)
\end{aligned} \tag{7}$$

$$\mathrm{Stree}_{23}(t) \wedge \neg\mathrm{Is\_leaf}(t) \;\rightarrow\; y \in_{23} \mathrm{Delete}_1(t,x) \leftrightarrow y \in_{23} t \wedge x \neq y \tag{8}$$

$$\mathrm{Stree}_{23}(t) \wedge \neg\mathrm{Is\_leaf}(t) \;\rightarrow\; \mathrm{Depth}(t) = \mathrm{Depth}\,\mathrm{Delete}_1(t,x). \tag{9}$$

Ak synmi uzla sú listy, tak odoberieme prvok pomocou $\mathrm{Delleaf}(x,ts)$. Ak nie a tak sa vnoríme do príslušného podstromu. Pri vynorení potrebujeme zavolať Delfixl alebo Delfixr, lebo môžeme dostať uzol s jedným synom. Ak sme sa vnorili do prvého alebo stredného podstromu, zavoláme Delfixl s nasledujúcim podstromom, ak sme sa vnorili do posledného, tak zavoláme Delfixr s predposledným podstromom. Tým môžeme vyrobiť uzol s jedným dvoma alebo troma synmi, pričom odstránime len prvok, ktorý sme chceli a zachováme hĺbky:

$$\begin{aligned}
&\mathrm{Delete}_1\left(\frac{\wedge}{t_1,ts},x\right) = \frac{\wedge}{\mathrm{Delleaf}((t_1,ts),x)} \;\leftarrow\; \mathrm{Is_leaf}(t_1)\\
&\mathrm{Delete}_1\left(\frac{\wedge}{t_1,ts},x\right) = \frac{\wedge}{\mathrm{Delfixl}(\mathrm{Delete}_1(t_1,x),t_2)} \;\leftarrow\\
&\qquad \neg\mathrm{Is_leaf}(t_1) \wedge ts = t_2,0 \wedge x \leq \mathrm{Max}_{23}(t_1)\\
&\mathrm{Delete}_1\left(\frac{\wedge}{t_1,ts},x\right) = \frac{\wedge}{\mathrm{Delfixr}(t_1,\mathrm{Delete}_1(t_2,x))} \;\leftarrow\\
&\qquad \neg\mathrm{Is_leaf}(t_1) \wedge ts = t_2,0 \wedge x > \mathrm{Max}_{23}(t_1)\\
&\mathrm{Delete}_1\left(\frac{\wedge}{t_1,ts},x\right) = \frac{\wedge}{\mathrm{Delfixl}(\mathrm{Delete}_1(t_1,x),t_2) \oplus (t_3,0)} \;\leftarrow\\
&\qquad \neg\mathrm{Is_leaf}(t_1) \wedge ts = t_2,t_3,0 \wedge x \leq \mathrm{Max}_{23}(t_1)
\end{aligned}$$

$$\mathrm{Delete}_1\left(\frac{\wedge}{t_1,ts},x\right)=\frac{\wedge}{t_1,\mathrm{Delfixl}\left(\mathrm{Delete}_1\left(t_2,x\right),t_3\right)}\leftarrow$$
$$\neg\mathrm{Is\_leaf}\left(t_1\right)\wedge ts=t_2,t_3,0\wedge x>\mathrm{Max}_{23}\left(t_1\right)\wedge x\leq\mathrm{Max}_{23}\left(t_2\right)$$
$$\mathrm{Delete}_1\left(\frac{\wedge}{t_1,ts},x\right)=\frac{\wedge}{t_1,\mathrm{Delfixr}\left(t_2,\mathrm{Delete}_1\left(t_3,x\right)\right)}\leftarrow$$
$$\neg\mathrm{Is\_leaf}\left(t_1\right)\wedge ts=t_2,t_3,0\wedge x>\mathrm{Max}_{23}\left(t_1\right)\wedge x>\mathrm{Max}_{23}\left(t_2\right).$$

Pre Delete takisto chceme štandardné vlastnosti:

$$\mathrm{Tree}_{23}\left(t\right)\;\rightarrow\;\mathrm{Tree}_{23}\,t\setminus\{x\} \tag{10}$$
$$\mathrm{Tree}_{23}\left(t\right)\;\rightarrow\;y\in_{23}t\setminus\{x\}\leftrightarrow y\in_{23}t\wedge x\neq y. \tag{11}$$

Tu vyriešime osobitne okrajové prípady, odoberanie z prázdneho stromu a z listu. Na ostané použijeme Delete_1. Ak dostaneme uzol, ktorý ma jediného syna, tak odstránime najvyšší uzol a koreň bude v jeho synovi. Obdobne ako pri vkladaní, tu je jediné miesto, kde sa mení hĺbka:

$$\bullet\setminus\{x\}=\bullet$$
$$[a]\setminus\{x\}=\bullet\;\leftarrow\;a=x$$
$$[a]\setminus\{x\}=[a]\;\leftarrow\;a\neq x$$
$$\frac{\wedge}{ts}\setminus\{x\}=ts_1\;\leftarrow\;\mathrm{Delete}_1\left(\frac{\wedge}{ts},x\right)=z\wedge\mathrm{Degree}(z)=1\wedge z=\frac{\wedge}{ts_1,0}$$
$$\frac{\wedge}{ts}\setminus\{x\}=z\;\leftarrow\;\mathrm{Delete}_1\left(\frac{\wedge}{ts},x\right)=z\wedge\mathrm{Degree}(z)>1.$$

Na záver ešte odobranie viacerých prvokv zo stromu:

$$\mathrm{Tree}_{23}\left(t\right)\;\rightarrow\;\mathrm{Tree}_{23}\,\mathrm{Delete\_list}(t,ls) \tag{12}$$
$$\mathrm{Tree}_{23}\left(t\right)\;\rightarrow\;x\in_{23}\mathrm{Delete\_list}(t,ls)\leftrightarrow x\in_{23}t\wedge xinls \tag{13}$$

Odoberať ich budeme podobne ako pri vkladaní:

$$\mathrm{Delete\_list}(t,0)=t$$
$$\mathrm{Delete\_list}(t,l,ls)=\mathrm{Delete\_list}(t\setminus\{l\},ls).$$

4 Červeno-čierne stromy

4.1 Farbené stromy (Colored trees)

4.1.1 Konštruktory a formát. Zavedieme *farbené stromy*. Budú to binárne stromy, pričom hodnoty budú uložené vo vnútorných uzloch. Listy nebudú obsahovať hodnoty, a budú mať vlastne význam ako ukončenie vetvy. Vo *farbených stromoch* má navyše každý vrchol priradenú farbu. Farba môže byť červená $\square$ alebo čierna $\blacksquare$. Zadefinujeme však aj farbu dvojitá čierna $\boxdot$, ktorú využijeme neskôr pri odoberaní prvkov z červeno-čierneho stromu. Farby aritmetizujeme do prirodzených čísel nasledovne:

$$\square = 0,0$$
$$\blacksquare = 1,0$$
$$\boxdot = 2,0.$$

Predikát $\mathrm{C}(c)$ je platný, ak c je farba. Definícia je zrejmá:

$$\mathrm{C}(\square)$$
$$\mathrm{C}(\blacksquare)$$
$$\mathrm{C}(\boxdot).$$

Ďalej môžeme prejsť k aritmetizácii stromov, potrebujeme na to tri konštruktory:

$$\bullet = 0,0$$
$$\odot = 1,0$$
$$\frac{x(c)}{l|r} = 2,x,c,l,r.$$

Prázdny strom alebo list budeme aritmetizovať konštruktorom $\bullet$ a tento bude mať stále čiernu farbu. Takisto pre odoberanie prvku potrebujeme dvojito čierny prázdny strom, ktorý aritmetizujeme konštruktorom $\odot$. Uzol bude aritmetizovaný $\frac{x(c)}{l|r}$, kde x je hodnota uložená v uzle, c je farba uzla, l je kód ľavého a r pravého podstromu. Rekurzívna definícia predikátu platiaceho pre kódy farbených stromov je:

$$\mathrm{Cbt}(\bullet)$$

$$\mathrm{Cbt}(\odot)$$
$$\mathrm{Cbt}\,\frac{x(c)}{l|r} \leftarrow \mathrm{N}(x) \wedge \mathrm{C}(c) \wedge \mathrm{Cbt}(l) \wedge \mathrm{Cbt}(r).$$

Stotožníme farbené stromy s ich kódmi a odteraz budeme vravieť farbený strom t namiesto kód farbeného stromu t.

4.1.2 Pomocné funkcie a predikáty. Pri farbených stromoch nás zaujímať čierna hĺbka stromu $\mathrm{Bl\_depth}\,(t)$. Bude to maximálny počet čiernych vrcholov na ceste od koreňa k listu. Dvojito čierny uzol budeme rátať dvakrát. Rekurzívna definícia:

$$\mathrm{Bl\_depth}\,(\bullet) = 0$$
$$\mathrm{Bl\_depth}\,(\odot) = 1$$
$$\mathrm{Bl\_depth}\,\frac{x(\blacksquare)}{l|r} = 1 + \max(\mathrm{Bl\_depth}\,(l), \mathrm{Bl\_depth}\,(r))$$
$$\mathrm{Bl\_depth}\,\frac{x(\square)}{l|r} = \max(\mathrm{Bl\_depth}\,(l), \mathrm{Bl\_depth}\,(r))$$
$$\mathrm{Bl\_depth}\,\frac{x(\boxdot)}{l|r} = 2 + \max(\mathrm{Bl\_depth}\,(l), \mathrm{Bl\_depth}\,(r)).$$

Pri niektorých ďalších algoritmoch nás niekedy bude zaujímať len farba koreňa podstromu, s ktorým budeme pracovať. Zavedieme teda takúto funkciu $\mathrm{Root\_col}\,(t)$. Defínicia:

$$\mathrm{Root\_col}\,(\bullet) = \blacksquare$$
$$\mathrm{Root\_col}\,(\odot) = \boxdot$$
$$\mathrm{Root\_col}\,\frac{x(c)}{l|r} = c.$$

Počet vrcholov stromu sa dá zistiť nasledovne:

$$\mathrm{Sz}(\bullet) = 0$$
$$\mathrm{Sz}\,\frac{x(c)}{l|r} = 1 + \mathrm{Sz}(l) + \mathrm{Sz}(r).$$

Predikát $x \in_{\mathrm{cbt}} t$ je platný, ak strom t obsahuje hodnotu x. Definujeme ho úplným prehľadávaním stromu:

$$x \in_{\mathrm{cbt}} \frac{y(c)}{l|r} \leftarrow x = y$$

$$
\begin{aligned}
&x \in_{\mathrm{cbt}} \frac{y(c)}{l|r} \leftarrow x \neq y \wedge x \in_{\mathrm{cbt}} l \\
&x \in_{\mathrm{cbt}} \frac{y(c)}{l|r} \leftarrow x \neq y \wedge \neg x \in_{\mathrm{cbt}} l \wedge x \in_{\mathrm{cbt}} r.
\end{aligned}
$$

Budeme potrebovať porovnávanie stromu t s nejakou hodnotou m. Pre predikát $t \succ m$ vyžadujeme aby platilo, že všetky prvky v strome t sú väčšie ako m. Teda

$$
\mathrm{Cbt}(t) \rightarrow t \succ m \leftrightarrow \forall x(x \in_{\mathrm{cbt}} t \rightarrow x > m). \tag{1}
$$

Rekurziou to vyriešime nasledovne

$$
\begin{aligned}
&\bullet \succ m \\
&\frac{y(c)}{l|r} \succ m \leftarrow y > m \wedge l \succ m \wedge r \succ m.
\end{aligned}
$$

Analogický prípad je $t \prec m$. Má platiť

$$
\mathrm{Cbt}(t) \rightarrow t \prec m \leftrightarrow \forall x(x \in_{\mathrm{cbt}} t \rightarrow x < m). \tag{2}
$$

Analogická je aj rekurzívna definícia

$$
\begin{aligned}
&\bullet \prec m \\
&\frac{y(c)}{l|r} \prec m \leftarrow y < m \wedge l \prec m \wedge r \prec m.
\end{aligned}
$$

Predikát $st \trianglelefteq t$ je platný, ak st je podstrom stromu t. Požadujeme aby platilo:

$$
\begin{aligned}
&\mathrm{Cbt}(st) \wedge \mathrm{Cbt}(t) \rightarrow \\
&st \trianglelefteq t \leftrightarrow st = t \vee \exists x \exists c \exists l \exists r \left(t = \frac{x(c)}{l|r} \wedge (st \trianglelefteq l \vee st \trianglelefteq r) \right).
\end{aligned} \tag{3}
$$

Rekurzívne ho definujeme takto:

$$
\begin{aligned}
&st \trianglelefteq \bullet \leftarrow st = \bullet \\
&st \trianglelefteq \frac{x(c)}{l|r} \leftarrow \frac{x(c)}{l|r} = st \\
&st \trianglelefteq \frac{x(c)}{l|r} \leftarrow \frac{x(c)}{l|r} \neq st \wedge st \trianglelefteq l \\
&st \trianglelefteq \frac{x(c)}{l|r} \leftarrow \frac{x(c)}{l|r} \neq st \wedge \neg st \trianglelefteq l \wedge st \trianglelefteq r.
\end{aligned}
$$

4.2 Červeno-čierne stromy

V nasledujúcom texte až do konca kapitoly budeme vychádzať z [1] až na vkladanie prvku. Tu využijeme [3].

4.2.1 Definícia. Červeno-čierne stromy sú farbené stromy, ktoré spĺňajú navyše nasledovné vlastnosti:

1. strom je binárnym prehľadávacím stromom,
2. každý uzol je buď červený alebo čierny,
3. koreň stromu a každý list je čierny,
4. ak je uzol červený, obaja jeho synovia sú čierni,
5. čierna hĺbka všetkých listov je rovnaká.

Podmienky 4 a 5 nám zabezpečujú vyváženosť stromu a to tak, že žiadna cesta nebude viac ako dvakrát tak dlhá ako ktorákoľvek iná.

4.2.2 Formalizácia definície. Najprv zadefinujeme predikát $\mathrm{Rbst}(t)$, platný pre všetky dobre skonštruované podstromy nejakého červeno-čierneho stromu:

$$
\begin{aligned}
&\mathrm{Rbst}(\bullet)\\
&\mathrm{Rbst}\,\frac{x(\blacksquare)}{l|r} \leftarrow\\
&\qquad \mathrm{Bl\_depth}\,(l) = \mathrm{Bl\_depth}\,(r) \wedge \mathrm{Rbst}(l) \wedge \mathrm{Rbst}(r) \wedge l \succ x \wedge r \prec x\\
&\mathrm{Rbst}\,\frac{x(\square)}{l|r} \leftarrow \mathrm{Root\_col}\,(l) = \blacksquare \wedge \mathrm{Root\_col}\,(r) = \blacksquare \wedge\\
&\qquad \mathrm{Bl\_depth}\,(l) = \mathrm{Bl\_depth}\,(r) \wedge \mathrm{Rbst}(l) \wedge \mathrm{Rbst}(r) \wedge l \succ x \wedge r \prec x.
\end{aligned}
$$

Predikát $\mathrm{Rb}(t)$ platí pre všetky červeno-čierne stromy, na rozdiel od predchádzajúceho predikátu vyžadujeme aby bol koreň čierny. Zadefinujeme ho pomocou $\mathrm{Rbst}(t)$:

$$
\begin{aligned}
&\mathrm{Rb}(\bullet)\\
&\mathrm{Rb}\,\frac{x(\blacksquare)}{l|r} \leftarrow \mathrm{Rbst}\,\frac{x(\blacksquare)}{l|r}.
\end{aligned}
$$

4.2.3 Prehľadávanie. Pre príslušnosť prvku $x \in_{\mathrm{rb}} t$ musí platiť

$$\mathrm{Rb}(t) \rightarrow x \in_{\mathrm{rb}} t \leftrightarrow x \in_{\mathrm{cbt}} t. \tag{1}$$

Keďže červeno-čierny strom je prehľadávacím stromom, testovanie na príslušnosť vieme urobiť binárnym prehľadávaním:

$$x \in_{\mathrm{rb}} \frac{y(c)}{l|r} \leftarrow x = y$$
$$x \in_{\mathrm{rb}} \frac{y(c)}{l|r} \leftarrow x < y \wedge x \in_{\mathrm{rb}} l$$
$$x \in_{\mathrm{rb}} \frac{y(c)}{l|r} \leftarrow x > y \wedge x \in_{\mathrm{rb}} r.$$

O minime platí:

$$\mathrm{Rb}(t) \rightarrow \mathrm{Minrb}(t) \in_{\mathrm{rb}} t \tag{2}$$
$$\mathrm{Rb}(t) \wedge x \in_{\mathrm{rb}} t \rightarrow \mathrm{Minrb}(t) \leq x. \tag{3}$$

Vďaka tomu, že strom t je usporiadaný, vieme, že minimum leží v najľavejšom vrchole:

$$\mathrm{Minrb}(\bullet) = 0$$
$$\mathrm{Minrb}\frac{x(c)}{l|r} = x \leftarrow l = \bullet$$
$$\mathrm{Minrb}\frac{x(c)}{l|r} = \mathrm{Minrb}(l) \leftarrow l \neq \bullet.$$

4.2.4 Insert. Vloženie prvku do červeno-čierneho stromu urobíme nasledovne. Najskôr binárnym prehľadávaním nájdeme miesto, kde by sa mal nachádzať vkladaný prvok. Tu pridáme červený vrchol s danou hodnotou a s dvoma listami. Toto nám však môže pokaziť štruktúru stromu, ak sme ho pridali pod červený uzol. Porušená teda môže byť len vlastnosť, že by po sebe nasledovali dva červené uzly.

Štruktúra sa nám nemusí pokaziť vždy, takže potrebujeme predikát, ktorý nám povie či ju daný uzol porušuje. Potrebujeme teda predikát, ktorý platí, ak je práve jeden zo synov červeného uzla červený:

$$t = \frac{x(c)}{l|r} \rightarrow \mathrm{Two\_red}(t) \leftrightarrow \mathrm{Root\_col}(l) = \square \wedge \mathrm{Root\_col}(r) = \blacksquare \vee \\ \mathrm{Root\_col}(l) = \blacksquare \wedge \mathrm{Root\_col}(r) = \square, \tag{1}$$

čo sa dá jednoducho zistiť takto:

$$\text{Two\_red}\ \frac{x(\square)}{l|r} \leftarrow \text{Root\_col}\,(l) = \square \wedge \text{Root\_col}\,(r) = \blacksquare$$
$$\text{Two\_red}\ \frac{x(\square)}{l|r} \leftarrow \text{Root\_col}\,(l) = \blacksquare \wedge \text{Root\_col}\,(r) = \square.$$

Tiež zadefinujeme predikát, ktorý overí, či sú splnené ostatné vlastnosti:

$$\text{Two\_red\_rbst}\ \frac{x(c)}{l|r} \leftarrow$$
$$\text{Two\_red}\ \frac{x(c)}{l|r} \wedge \text{Rbst}(l) \wedge \text{Rbst}(r) \wedge \text{Bl\_depth}\,(l) = \text{Bl\_depth}\,(r).$$

Samotnú opravu štruktúry však musíme riešiť z pozície jeho otca. Budeme ju riešiť v dvoch samostatných funkciách, podľa toho či je poškodený ľavý alebo pravý podstrom. Teda ak štruktúru porušuje koreň alebo ak máme dobrý podstrom, nič nebudeme opravovať. Ak je ľavý podstrom porušený, výsledkom úpravy bude platný podstrom, ktorý bude obsahovať tie isté prvky a bude mať rovnakú hĺbku. Formálne zapísané:

$$\text{Rbst}(t) \wedge t \neq \bullet \vee \text{Two\_red\_rbst}\,(t) \rightarrow \text{Insfixl}(t) = t \tag{2}$$

$$t = \frac{x(\blacksquare)}{l|r} \wedge \text{Rbst}(r) \wedge \text{Two\_red\_rbst}\,(l) \wedge \text{Bl\_depth}\,(l) = \text{Bl\_depth}\,(r) \rightarrow \\ \text{Rbst}\,\text{Insfixl}(t) \tag{3}$$

$$t = \frac{x(\blacksquare)}{l|r} \wedge \text{Rbst}(r) \wedge \text{Two\_red\_rbst}\,(l) \wedge \text{Bl\_depth}\,(l) = \text{Bl\_depth}\,(r) \rightarrow \\ \text{Bl\_depth}\,(t) = \text{Bl\_depth}\ \text{Insfixl}(t) \tag{4}$$

$$t = \frac{x(\blacksquare)}{l|r} \wedge \text{Two\_red\_rbst}\,(l) \rightarrow x \in_{\text{rb}} t \leftrightarrow x \in_{\text{rb}} \text{Insfixl}(t). \tag{5}$$

Opravovať budeme teda dva prípady, podľa pozície druhého červeného uzla. Budeme to riešiť pomocou rotácií [1]:

- $\dfrac{x_1(\blacksquare)}{\frac{x_2(\square)}{\frac{x_3(\square)}{l_3|r_3}|r_2}|r_1}$ – druhá klauzula definície: Urobíme rotáciu vpravo okolo uzla x_1, pričom, prefarbíme uzol x_3 na čierno.
- $\dfrac{x_1(\blacksquare)}{\frac{x_2(\square)}{l_2|\frac{x_4(\square)}{l_4|r_4}}|r_1}$ – tretia klauzula: Tu urobíme najskôr rotáciu vľavo okolo uzla x_2 a potom vpravo okolo x_1.

Ľahko vidieť, že sme zachovali všetky prvky a tiež, že čiere hĺbky všetkých podstromov sa zachovali:

$$
\begin{aligned}
&\mathrm{Insfixl}\,\frac{x_1(c)}{l_1|r_1} = \frac{x_1(c)}{l_1|r_1} \leftarrow \neg\mathrm{Two\_red}\,(l_1)\\
&\mathrm{Insfixl}\,\frac{x_1(c)}{l_1|r_1} = \frac{x_2(\square)}{\frac{x_3(\blacksquare)}{l_3|r_3}|\frac{x_1(\blacksquare)}{r_2|r_1}} \leftarrow \mathrm{Two\_red}\,(l_1) \wedge c = \blacksquare \wedge l_1 = \frac{x_2(\square)}{l_2|r_2} \wedge\\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{Root\_col}\,(l_2) = \square \wedge l_2 = \frac{x_3(\square)}{l_3|r_3}\\
&\mathrm{Insfixl}\,\frac{x_1(c)}{l_1|r_1} = \frac{x_4(\square)}{\frac{x_2(\blacksquare)}{l_2|l_4}|\frac{x_1(\blacksquare)}{r_4|r_1}} \leftarrow \mathrm{Two\_red}\,(l_1) \wedge c = \blacksquare \wedge l_1 = \frac{x_2(\square)}{l_2|r_2} \wedge\\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{Root\_col}\,(l_2) = \blacksquare \wedge r_2 = \frac{x_4(\square)}{l_4|r_4}.
\end{aligned}
$$

Obdobne riešime aj prípad, ak pravý podstrom porušuje štruktúru:

$$
\begin{aligned}
&\mathrm{Insfixr}\,\frac{x_1(c)}{l_1|r_1} = \frac{x_1(c)}{l_1|r_1} \leftarrow \neg\mathrm{Two\_red}\,(r_1)\\
&\mathrm{Insfixr}\,\frac{x_1(c)}{l_1|r_1} = \frac{x_3(\square)}{\frac{x_1(\blacksquare)}{l_1|l_3}|\frac{x_2(\blacksquare)}{r_3|r_2}} \leftarrow \mathrm{Two\_red}\,(r_1) \wedge c = \blacksquare \wedge r_1 = \frac{x_2(\square)}{l_2|r_2} \wedge\\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{Root\_col}\,(l_2) = \square \wedge l_2 = \frac{x_3(\square)}{l_3|r_3}\\
&\mathrm{Insfixr}\,\frac{x_1(c)}{l_1|r_1} = \frac{x_2(\square)}{\frac{x_1(\blacksquare)}{l_1|l_2}|\frac{x_4(\blacksquare)}{l_4|r_4}} \leftarrow \mathrm{Two\_red}\,(r_1) \wedge c = \blacksquare \wedge r_1 = \frac{x_2(\square)}{l_2|r_2} \wedge\\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{Root\_col}\,(l_2) = \blacksquare \wedge r_2 = \frac{x_4(\square)}{l_4|r_4}.
\end{aligned}
$$

Pri vkladaní prvku do podstromu vyžadujeme, aby vo výslednom strome boli všetky prvky pôvodného stromu a tiež vkladaný prvok a aby sme zachovali výšku. Dôležité je tiež, že pri vkladaní do podstromu s čiernym koreňom dostaneme dobrý podstrom. Ak vkladáme do červeného, výsledok môže byť aj strom, ktorého koreň porušuje štruktúru. Táto vlastnosť je dôležitá aby sme po skončení vkladania mali platný podstrom. Tieto požiadavky sformulujeme nasledovne:

$$\mathrm{Rbst}(t) \rightarrow y \in_{\mathrm{rb}} \mathrm{Insert}_1\,(t,x) \leftrightarrow y \in_{\mathrm{rb}} t \vee y = x \tag{6}$$

$$\mathrm{Rbst}(t) \rightarrow \mathrm{Bl\_depth}\,\mathrm{Insert}_1\,(t,x) = \mathrm{Bl\_depth}\,(t) \tag{7}$$

$$\begin{aligned}\mathrm{Rbst}(t) \rightarrow &\,\mathrm{Rbst}\,\mathrm{Insert}_1\,(t,x) \vee\\ &\mathrm{Root\_col}\,(t) = \square \wedge \mathrm{Two\_red\_rbst}\,\mathrm{Insert}_1\,(t,x).\end{aligned} \tag{8}$$

Vkladať prvok budeme binárnym prehľadávaním, teda ak je prvok menší, vložíme ho do ľavého, a ak väčší tak do pravého podstromu. Potom na celý strom zavoláme funkciu Insfixl alebo Insfixr podľa toho, kam sme prvok vkladali. Vkladanie do listu spočíva v tom, že vytvoríme nový červený uzol:

$$\begin{aligned}
&\mathrm{Insert}_1\,(\bullet,x) = \frac{x(\square)}{\bullet|\bullet}\\
&\mathrm{Insert}_1\left(\frac{y(c)}{l|r},x\right) = \mathrm{Insfixl}\,\frac{y(c)}{\mathrm{Insert}_1\,(l,x)|r} \leftarrow x<y\\
&\mathrm{Insert}_1\left(\frac{y(c)}{l|r},x\right) = \frac{y(c)}{l|r} \leftarrow x=y\\
&\mathrm{Insert}_1\left(\frac{y(c)}{l|r},x\right) = \mathrm{Insfixr}\,\frac{y(c)}{l|\mathrm{Insert}_1\,(r,x)} \leftarrow x>y.
\end{aligned}$$

Predošlá funkcia nám mohla vrátiť strom s červeným koreňom, teda ešte potrebujeme funkciu, ktorá spĺňa

$$\mathrm{Rb}(t) \to y \in_{\mathrm{rb}} t \cup \{x\} \leftrightarrow y \in_{\mathrm{rb}} t \vee y = x \tag{9}$$

$$\mathrm{Rb}(t) \to \mathrm{Rb}\, t \cup \{x\}. \tag{10}$$

A v nej vlastne len prefarbíme vrchol na čierno, ak bol červený. Toto je jediný spôsob ako môžeme zväčšiť čiernu hĺbku stromu:

$$t \cup \{x\} = \frac{y(\blacksquare)}{l_1|r_1} \leftarrow \mathrm{Insert}_1\,(t,x) = \frac{y(c)}{l_1|r_1}.$$

Pre vkladanie zoznamu prvkov musí platiť:

$$\mathrm{Rb}(t) \to \mathrm{Rb}\,\mathrm{Insert\_list}(t,ls) \tag{11}$$

$$\mathrm{Rb}(t) \to x \in_{\mathrm{rb}} \mathrm{Insert\_list}(t,ls) \leftrightarrow x \in_{\mathrm{rb}} t \vee x\, \varepsilon\, ls. \tag{12}$$

Postupne po jednom teda budeme prvky pridávať:

$$\begin{aligned}
&\mathrm{Insert_list}(t,0) = t\\
&\mathrm{Insert_list}(t,l,ls) = \mathrm{Insert_list}(t \cup \{l\},ls)
\end{aligned}$$

4.2.5 Delete. Vymazávanie urobíme podobne ako vkladanie. Problém však nastáva, keď odstraňujeme čierny uzol, lebo listy vo vetve pod ním budú mať o jedno menšiu hĺbku. Tu využijeme farbu $\boxdot$, ktorú sme si na začiatku zadefinovali. Strom bude mať maximálne jeden takýto uzol, a vďaka tomu, že do čiernej hĺbky sa ráta dvakrát, ostanú zachované všetky ostatné vlastnosti červeno-čiernych stromov. Takýto uzol bude vlastne tiež akýsi ukazovateľ na miesto, kde musíme strom upravovať.

Zadefinujeme predikát, ktorý nám povie, či koreň má farbu $\boxdot$ a či jeho synovia sú platné podstromy s rovnakou hĺbkou:

$$\begin{aligned}
&\mathrm{Havedbl}(\odot)\\
&\mathrm{Havedbl}\,\frac{x(c)}{l|r} \leftarrow c = \boxdot \wedge \mathrm{Rbst}(l) \wedge \mathrm{Rbst}(r) \wedge \mathrm{Bl_depth}\,(l) = \mathrm{Bl_depth}\,(r).
\end{aligned}$$

Niekedy budeme chcieť meniť farbu koreňa stromu bez toho, aby nás zaujímal zvyšný obsah uzla. Zavedieme operáciu $\mathrm{Inccol}(t)$, ktorá „inkrementuje“ farbu koreňa stromu t. Pod inkrementáciou farby budeme chápať zmenu farby z $\blacksquare$ na $\boxdot$ alebo z $\square$ na $\blacksquare$. Ak t je $\bullet$ tak ho zmeníme na $\odot$. Môžeme teda povedať:

$$\mathrm{Rbst}(t) \rightarrow x \in_{\mathrm{rb}} \mathrm{Inccol}(t) \leftrightarrow x \in_{\mathrm{rb}} t \tag{1}$$

$$\mathrm{Rbst}(t) \rightarrow \mathrm{Bl\_depth}\,\mathrm{Inccol}(t) = \mathrm{Bl\_depth}\,(t) + 1 \tag{2}$$

$$\mathrm{Rbst}(t) \rightarrow \mathrm{Havedbl}\,\mathrm{Inccol}(t) \vee \mathrm{Rb}\,\mathrm{Inccol}(t). \tag{3}$$

Samotná funkcia je jednoduchá:

$$\begin{aligned}
&\mathrm{Inccol}(\bullet) = \odot\\
&\mathrm{Inccol}\,\frac{x(\square)}{l|r} = \frac{x(\blacksquare)}{l|r}\\
&\mathrm{Inccol}\,\frac{x(\blacksquare)}{l|r} = \frac{x(\boxdot)}{l|r}.
\end{aligned}$$

Obdobne budeme potrebovať aj „dekrementovať“ farbu, postačí nám zmena z $\boxdot$ na $\blacksquare$ a z $\odot$ na $\bullet$:

$$\mathrm{Havedbl}(t) \rightarrow \mathrm{Rbst}\,\mathrm{Deccol}(t) \tag{4}$$

$$\mathrm{Havedbl}(t) \rightarrow x \in_{\mathrm{rb}} \mathrm{Deccol}(t) \leftrightarrow x \in_{\mathrm{rb}} t \tag{5}$$

$$\mathrm{Havedbl}(t) \rightarrow \mathrm{Bl\_depth}\,\mathrm{Deccol}(t) + 1 = \mathrm{Bl\_depth}\,(t). \tag{6}$$

Definícia:

$$\begin{aligned}
&\mathrm{Deccol}(\odot) = \bullet\\
&\mathrm{Deccol}\,\frac{x(\boxdot)}{l|r} = \frac{x(\blacksquare)}{l|r}.
\end{aligned}$$

Úpravu budeme znova riešiť ako pri vkladaní z pozície otca a rovnako rozlíšime, či je dvojito čierny ľavý alebo pravý syn. Ak dostaneme strom s ľavým dvojito čiernym synom, chceme aby po sme po úprave dostali buď platný podstrom alebo sa presunula dvojitá čierna farba na otca, zachovať všetky prvky a tiež výšky:

$$\mathrm{Rbst}(t) \rightarrow \mathrm{Delfixlrb}(t) = t \tag{7}$$

$$\begin{aligned} t = \frac{x(c)}{l|r} \wedge \mathrm{Havedbl}(l) \wedge \mathrm{Rbst}(r) \wedge \mathrm{Bl\_depth}\,(l) = \mathrm{Bl\_depth}\,(r) \rightarrow \\ \mathrm{Havedbl}\,\mathrm{Delfixlrb}(t) \vee \mathrm{Rbst}\,\mathrm{Delfixlrb}(t) \end{aligned} \tag{8}$$

$$\begin{aligned} t = \frac{x(c)}{l|r} \wedge \mathrm{Havedbl}(l) \wedge \mathrm{Rbst}(r) \wedge \mathrm{Bl\_depth}\,(l) = \mathrm{Bl\_depth}\,(r) \rightarrow \\ \mathrm{Bl\_depth}\;\mathrm{Delfixlrb}(t) = \mathrm{Bl\_depth}\,(t) \end{aligned} \tag{9}$$

$$\begin{aligned} t = \frac{x(c)}{l|r} \wedge \mathrm{Havedbl}(l) \wedge \mathrm{Rbst}(r) \wedge \mathrm{Bl\_depth}\,(l) = \mathrm{Bl\_depth}\,(r) \rightarrow \\ x \in_{\mathrm{rb}} t \leftrightarrow x \in_{\mathrm{rb}} \mathrm{Delfixlrb}(t). \end{aligned} \tag{10}$$

Ak budeme predpokladať, že l je dvojito čierny, môžeme rozlíšiť štyri prípady (ak nás bude zaujímať farba koreňa nejakého podstromu, napíšeme tc namiesto t):

- $\frac{x(c)}{l|\frac{x_2(\square)}{l_2|r_2}}$ – druhá klauzula definície: Vykonáme rotáciu vľavo okolo x a na ľavý podstrom zavoláme znova funkciu Delfixl. Takto prevedieme tento prípad na niektorý z nasledujúcich.
- $\frac{x(c)}{l|\frac{x_2(\blacksquare)}{l_2\blacksquare|r_2\blacksquare}}$ – tretia klauzula: Prefarbíme x_2 na červeno, inkrementujeme farbu x a dekrementujeme farbu l.
- $\frac{x(c)}{l|\frac{x_2(\blacksquare)}{l_2|r_2\square}}$ – štvrtá klauzula: Urobíme rotáciu vľavo okolo x, dekrementujeme farbu l, inkrementujeme r_2 a x_2 dostane farbu pôvodného koreňa.
- $\frac{x(c)}{l|\frac{x_2(\blacksquare)}{\frac{x_3(\square)}{l_3|r_3}|r_2\blacksquare}}$ – piata klauzula: Urobíme rotáciu vpravo okolo x_2, potom vľavo okolo x, x_3 prefarbíme na farbu x, x na čierno a dekrementujeme farbu l.

Je dobré si uvedomiť, že jediný prípad, kedy môžeme dostať strom s dvojito čiernym koreňom, je prípad dva a to len vtedy, ak farba x bola čierna. Z toho tiež ďalej vyplynie, že ak sme prípad jedna previedli na prípad dva, tak výsledkom bude dobrý podstrom. Definícia:

$$\mathrm{Delfixlrb}\,\frac{x(c)}{l|r} = \frac{x(c)}{l|r} \leftarrow \mathrm{Root\_col}\,(l) \neq \boxdot$$

$$\begin{aligned}
&\text{Delfixlrb}\,\frac{x(c)}{l|r} = \frac{x_2(\blacksquare)}{\text{Delfixlrb}\,\frac{x(\square)}{l|l_2}|r_2} \leftarrow \text{Root_col}\,(l) = \boxdot \wedge r = \frac{x_2(\square)}{l_2|r_2} \\
&\text{Delfixlrb}\,\frac{x(c)}{l|r} = \text{Inccol}\,\frac{x(c)}{\text{Deccol}(l)|\frac{x_2(\square)}{l_2|r_2}} \leftarrow \text{Root_col}\,(l) = \boxdot \wedge r = \frac{x_2(\blacksquare)}{l_2|r_2} \wedge \\
&\qquad\qquad \text{Root_col}\,(r_2) = \blacksquare \wedge \text{Root_col}\,(l_2) = \blacksquare \\
&\text{Delfixlrb}\,\frac{x(c)}{l|r} = \frac{x_2(c)}{\frac{x(\blacksquare)}{\text{Deccol}(l)|l_2}|\text{Inccol}(r_2)} \leftarrow \text{Root_col}\,(l) = \boxdot \wedge r = \frac{x_2(\blacksquare)}{l_2|r_2} \wedge \\
&\qquad\qquad \text{Root_col}\,(r_2) = \square \\
&\text{Delfixlrb}\,\frac{x(c)}{l|r} = \frac{x_3(c)}{\frac{x(\blacksquare)}{\text{Deccol}(l)|l_3}|\frac{x_2(\blacksquare)}{r_3|r_2}} \leftarrow \text{Root_col}\,(l) = \boxdot \wedge r = \frac{x_2(\blacksquare)}{l_2|r_2} \wedge \\
&\qquad\qquad \text{Root_col}\,(r_2) = \blacksquare \wedge \text{Root_col}\,(l_2) = \square \wedge l_2 = \frac{x_3(\square)}{l_3|r_3}.
\end{aligned}$$

A analogicky postupujeme v situácii, keď je dvojito čierny pravý podstrom:

$$\begin{aligned}
&\text{Delfixrrb}\,\frac{x(c)}{l|r} = \frac{x(c)}{l|r} \leftarrow \text{Root_col}\,(r) \neq \boxdot \\
&\text{Delfixrrb}\,\frac{x(c)}{l|r} = \frac{x_2(\blacksquare)}{l_2|\text{Delfixrrb}\,\frac{x(\square)}{r_2|r}} \leftarrow \text{Root_col}\,(r) = \boxdot \wedge l = \frac{x_2(\square)}{l_2|r_2} \\
&\text{Delfixrrb}\,\frac{x(c)}{l|r} = \text{Inccol}\,\frac{x(c)}{\frac{x_2(\square)}{l_2|r_2}|\text{Deccol}(r)} \leftarrow \text{Root_col}\,(r) = \boxdot \wedge l = \frac{x_2(\blacksquare)}{l_2|r_2} \wedge \\
&\qquad\qquad \text{Root_col}\,(l_2) = \blacksquare \wedge \text{Root_col}\,(r_2) = \blacksquare \\
&\text{Delfixrrb}\,\frac{x(c)}{l|r} = \frac{x_2(c)}{\text{Inccol}(l_2)|\frac{x(\blacksquare)}{r_2|\text{Deccol}(r)}} \leftarrow \text{Root_col}\,(r) = \boxdot \wedge l = \frac{x_2(\blacksquare)}{l_2|r_2} \wedge \\
&\qquad\qquad \text{Root_col}\,(l_2) = \square \\
&\text{Delfixrrb}\,\frac{x(c)}{l|r} = \frac{x_3(c)}{\frac{x_2(\blacksquare)}{l_2|l_3}|\frac{x(\blacksquare)}{r_3|\text{Deccol}(r)}} \leftarrow \text{Root_col}\,(r) = \boxdot \wedge l = \frac{x_2(\blacksquare)}{l_2|r_2} \wedge \\
&\qquad\qquad \text{Root_col}\,(l_2) = \blacksquare \wedge \text{Root_col}\,(r_2) = \square \wedge r_2 = \frac{x_3(\square)}{l_3|r_3}.
\end{aligned}$$

Ešte pred funkciou mazania prvku si zadefinujeme pomocnú funkciu, ktorá inkrementuje farbu stromu, ak je druhý argument $\square$, inak vráti neporušený strom:

$$\text{Inc}(t, \square) = t$$

$\mathrm{Inc}(t, \blacksquare) = \mathrm{Inccol}(t)$.

Budeme požadovať aby po odstránení prvku sme dostali platný podstrom, ktorý bude obsahovať všetky hodnoty pôvodného okrem odstraňovanej a chceme tiež zachovať čiernu hĺbku:

$$\mathrm{Rbst}(t) \rightarrow \mathrm{Rbst}\,\mathrm{Delete}_1\,(t,x) \tag{11}$$

$$\mathrm{Rbst}(t) \rightarrow \mathrm{Bl\_depth}\,\mathrm{Delete}_1\,(t,x) = \mathrm{Bl\_depth}\,(t) \tag{12}$$

$$\mathrm{Rbst}(t) \rightarrow y \in_{\mathrm{rb}} \mathrm{Delete}_1\,(t,x) \leftrightarrow y \in_{\mathrm{rb}} t \wedge x \neq y. \tag{13}$$

Pri vymazávaní hodnoty x zo stromu t najskôr binárnym prehľadávaním nájdeme uzol, v ktorom je uložená daná hodnota. Ak ani jeden z potomkov tohto uzla nie je list, tak zoberieme minimum m pravého syna, vložíme ho do tohto uzla. Ďalej budeme pokračovať v rekurzii v tomto synovi, budeme však mazať m. Týmto sme dosiahli, že keď budeme chcieť odstrániť nejaký podstrom, tak jeden jeho syn bude list, teda druhý podstrom môžeme „nadpojiť" na jeho miesto. Ak sme odstraňovali červený uzol, výsledný strom bude platný červeno-čierny strom. Ak sme odstraňovali čierny uzol, tak inkrementujeme farbu pripájaného podstromu. Definícia:

$$\begin{aligned}
&\mathrm{Delete}_1\,(\bullet,x) = \bullet\\
&\mathrm{Delete}_1\left(\frac{y(c)}{l|r},x\right) = \mathrm{Delfixlrb}\,\frac{y(c)}{\mathrm{Delete}_1\,(l,x)|r} \leftarrow x < y\\
&\mathrm{Delete}_1\left(\frac{y(c)}{l|r},x\right) = \mathrm{Delfixrrb}\,\frac{y(c)}{l|\mathrm{Delete}_1\,(r,x)} \leftarrow x > y\\
&\mathrm{Delete}_1\left(\frac{y(c)}{l|r},x\right) = \mathrm{Delfixrrb}\,\frac{m(c)}{l|\mathrm{Delete}_1\,(r,m)} \leftarrow\\
&\qquad x = y \wedge l \neq \bullet \wedge r \neq \bullet \wedge \mathrm{Minrb}(r) = m\\
&\mathrm{Delete}_1\left(\frac{y(c)}{l|r},x\right) = \mathrm{Inc}(l,c) \leftarrow x = y \wedge l \neq \bullet \wedge r = \bullet\\
&\mathrm{Delete}_1\left(\frac{y(c)}{l|r},x\right) = \mathrm{Inc}(r,c) \leftarrow x = y \wedge l = \bullet.
\end{aligned}$$

Štandardná vlastnosť pre odstraňovanie prvku:

$$\mathrm{Rb}(t) \rightarrow y \in_{\mathrm{rb}} t \setminus \{x\} \leftrightarrow y \in_{\mathrm{rb}} t \wedge y \neq x \mathrm{Rb}(t) \rightarrow \mathrm{Rb} t \setminus \{x\}. \tag{14}$$

Takisto ako pri vkladaní, mohli sme dostať strom, ktorý nemá čierny koreň, tu však môžeme dostať koreň s dvojite čiernou farbou. Takýto koreň prefarbíme na čierno, čím zmenšíme hĺbku celého stromu:

$$t \setminus \{x\} = \bullet \leftarrow t = \bullet$$

$$t \setminus \{x\} = \frac{y(\blacksquare)}{l|r} \leftarrow t \neq \bullet \wedge \mathrm{Delete}_1\,(t,x) = \frac{y(c)}{l|r}.$$

Pre odstraňovanie zoznamu prvkov platí:

$$\mathrm{Rb}(t) \rightarrow \mathrm{Rb}\,\mathrm{Delete\_list}(t,ls) \tag{15}$$

$$\mathrm{Rb}(t) \rightarrow x \in_{\mathrm{rb}} \mathrm{Delete\_list}(t,ls) \leftrightarrow x \in_{\mathrm{rb}} t \wedge x \not\in ls. \tag{16}$$

Znova to vyriešime postupným odoberaním:

$$\mathrm{Delete\_list}(t,0) = t$$
$$\mathrm{Delete\_list}(t,l,ls) = \mathrm{Delete\_list}(t \setminus \{l\},ls).$$

5 Záver

Ciele týkajúce sa implementácie vyvážených stromov sme splnili. Implementovoli sme uvedené štruktúry a operácie na nich. Keďže uzly stromov v imperatívnej definícii obsahovali smerníky nahor, vyskytli sa problémy, keďže v deklaratívnom programovaní sa nič podobné ako smerníky nepoužíva. Takisto nepoznáme nejaký všeobecný spôsob, ako takéto situácie riešiť. V našom prípade prípade však bolo riešenie v celku jednoduché: ak sme potrebovali pracovať s otcom nejakého uzla, tak sme len počkali, kým sa vynoríme z rekurzie na danú úroveň.

Na rozdiel od pôvodného cieľa, vkladanie a odoberanie prvku v 2-3 stromoch sme urobili v časovej zložitosti $O(\log^2 n)$. Bol to ústupok lepšej prehľadnosti a čitateľnosti implementácie. Principiálne už nie je ťažké upraviť túto implementáciu tak, aby spĺňala aj požadovanú časovú zložitosť.

Samotná špecifikácia operácií vkladania a odoberania prvku je jednoduchá, dôležité však bolo špecifikovať pomocné funkcie, ktoré využívajú. Toto sme splnili a pripravili sme dobré východisko na ich verifikáciu.

Literatúra

[1] Procházka, Juraj. *Algoritmy a dátové štruktúry.* Vysokoškolské skriptá. FMFI UK, Bratislava, 1997.

[2] Ďuriš, Pavol. *Tvorba efektívnych algoritmov.* Vysokoškolské skriptá. FMFI UK, Bratislava, 1996.

[3] Okasaki, Chris. *Purely Functional Data Structures.* Cambridge University Press, 1998.

[4] Komara, Ján, Voda, Pavol. *Metamathematics of Computer Programming.* Vysokoškolské skriptá. FMFI UK, Bratislava, 2001.

[5] Rudolf Bayer, Edward M. McCreight. *Organization and Maintenance of Large Ordered Indices.* Acta Informatica 1: 173-189 (1972)