FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Semestrální práce z předmětu KIV/PRO

Vykreslování stromů

Patrik Harag

harag@students.zcu.cz

(A15B0034P)

26. listopadu 2016

Obsah

Kapitola 1

Zadání

Mnoho aplikací vyžaduje vykreslování obrázků stromů. Stromové diagramy jsou běžně používané pro vizualizaci adresářových struktur, rodokmenů nebo třeba abstraktních syntaktických stromů. Rozlišují se dva základní druhy stromů:

- *Zakořeněný strom* představuje hierarchii vycházející z jediného vrcholu nazývaného kořen. Diagram by měl reflektovat tuto hierarchii včetně pořadí, ve kterém se vyskytují jednotlivé vrcholy. Například rodokmeny jsou zakořeněné stromy a vrcholy představující sourozence jsou vykreslovány zleva doprava podle data narození.
- *Volný strom* nemá žádný kořen ani hierarchickou strukturu. K vykreslování se tedy přistupuje zcela odlišným způsobem.

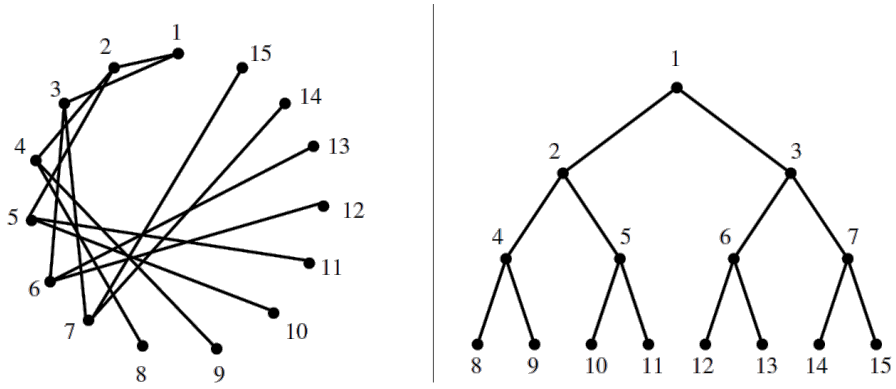
V této práci se budeme zabývat výhradně vykreslováním zakořeněných stromů.

Vstup

Strom T, jenž je grafem bez cyklů. Rozměry vrcholů se neuvažují.

Výstup

Hezký obrázek stromu T.



Obrázek 1.1: Příklad vstupu a výstupu. Zdroj: [Ski09]

Kapitola 2

Existující metody

Existují dva základní způsoby vizualizace zakořeněných stromů:

- *Hierarchické rozložení* – kořen je umístěn do středu horního okraje obrázku, strom roste směrem dolů. Alternativně může strom růst odspoda nahoru nebo z jedné boční strany na druhou.
- *Středové rozložení* – kořen je umístěn do středu obrázku, strom roste všemi směry. Toto rozložení je prostorově úspornější.

Výběr záleží především na konkrétní aplikaci. My se ale budeme zabývat výhradně standardním hierarchickým rozložením.

2.1 Jednoduchý rekurzivní algoritmus

Algoritmus umístí kořen doprostřed horního okraje obrázku a obrázek vertikálně rozdělí na n částí, kde n je počet podstromů. Podstromy se rekurzivně vykreslí. Kořen podstromu je vždy umístěn do středu šířky, kterou zabírá celý jeho podstrom.

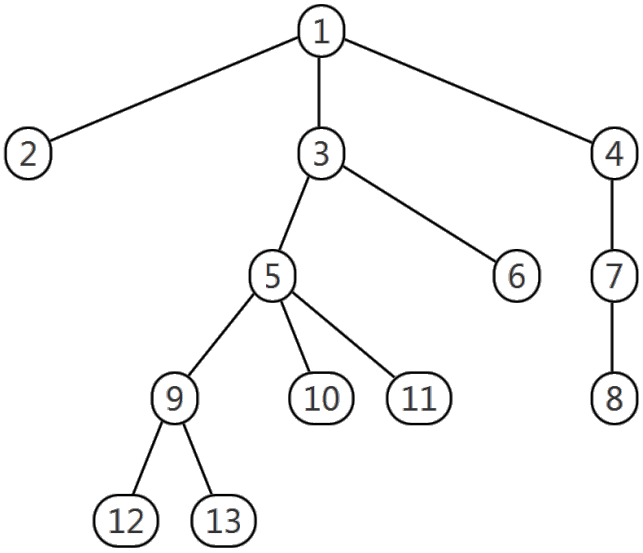
Ovšem tento algoritmus může vytvořit rozsáhlé prázdné plochy, jak je vidět na Obrázku 2.1. To je dáno skutečností, že jsou sousední podstromy vykresleny vždy vedle sebe v pomyslném obdélníku a nemohou do sebe navzájem zasahovat. Tento problém nevznikne u vyvážených stromů, jako je například strom na Obrázku 1.1.

2.2 Reingold-Tilfordův algoritmu

Reingold-Tilfordův algoritmus v podobě, v jaké byl představen [MR], byl původně určen především k vykreslování binárních stromů. Nicméně je možné ho zobecnit, aby fungoval pro libovolný n-nární strom.

Algoritmus spočívá v rekurzivní stavbě stromu odspoda. Jednotlivé podstromy se přiblíží a následně se nad ně vloží rodič zarovnaný na střed. Podstromy se takto postupně budují, dokud nedojde k vytvoření celého stromu.

Důležité vlastnosti tohoto algoritmu jsou, že se snaží strom vykreslit s co nejmenší šířkou a zachovává symetričnost. Pracuje v lineárním čase.



Obrázek 2.1: Příklad stromu nakresleného jednoduchým rozmísťovacím algoritmem. Nevýhody tohoto algoritmu jsou jasně viditelné.

2.3 Walkerův algoritmus

Walkerův algoritmus [Wal90] vychází z Reingold-Tilfordova algoritmu, ale zobecňuje jej pro použití na n-nárních stromech.

Algoritmus je dvouprůchodový. V prvním průchodu metodou *postorder* jsou každému vrcholu přiřazeny dočasné souřadnice na ose x. V druhém průchodu metodou *preorder* jsou tyto souřadnice ještě přepočítány.

Walkerův algoritmus byl později vylepšen, aby pracoval v lineárním čase [CB02].

Kapitola 3

Navržené řešení

3.1 Základní algoritmus

Rozhodl jsem se navrhnout své vlastní řešení. Algoritmus funguje tak, že prochází strom metodou *postorder* a přitom určuje pozice jednotlivých vrcholů i celých podstromů.

Každý podstrom je nejprve sestaven samostatně a pro každou jeho úroveň jsou určeny levé a pravé okraje. Jak jsou po sobě, zleva doprava, vytvářeny jednotlivé větve podstromu, je na základě levých okrajů všech jeho úrovní určena minimální velikost, o kolik se musí větev posunout směrem doprava. Poté se všechny vrcholy o tuto velikost posunou a dojde k přepočítání okrajů podstromu. Když jsou všechny větve podstromu zpracovány, zarovná se nad nimi jejich předek.

3.2 Vylepšený algoritmus

Výkon algoritmu popsaného výše kazí skutečnost, že je nutné opakovaně posouvat celé podstromy. U větších stromů by to mělo výrazný vliv na výkon.

Řešením je neposouvat celý podstrom ihned, ale jen uložit velikost posunutí do kořenu podstromu. Po dokončení standardního průchodu se celý strom projde ještě jednou a velikost posunutí se pro každý vrchol agreguje.

3.2.1 Pseudokód

```
void make(tree) {
    makeSubtree(tree.getRoot(), 0);
    finalizeMove(tree.getRoot(), 0);
}

Bounds makeSubtree(node, level) {
    if (node nemá potomky) {
        nastavit node.x = 0, node.y = level * verticalSize
        return okraje vrcholu

    } else {
        bounds = struktura pro uložení okrajů tohoto podstromu
        for (následníci n vrcholu node) {
            branch_bounds = makeSubtree(n, level + 1);
            výpočet velikosti posunu větve n směrem doprava na základě
                bounds a branch_bounds
            nastavení informace o posunutí n
            přepočítání okrajů podstromu (bounds)
        }

        nastavit node.x = vycentrování node nad jeho potomky,
                 node.y = level * verticalSize

        přidání okrajů node do bounds
        return bounds
    }
}

void finalizeMove(node, totalOffset) {
    totalOffset += node.offset

    for (následníci n vrcholu node) {
        n.x += totalOffset;
        finalizeMove(n, totalOffset);
    }
}
```

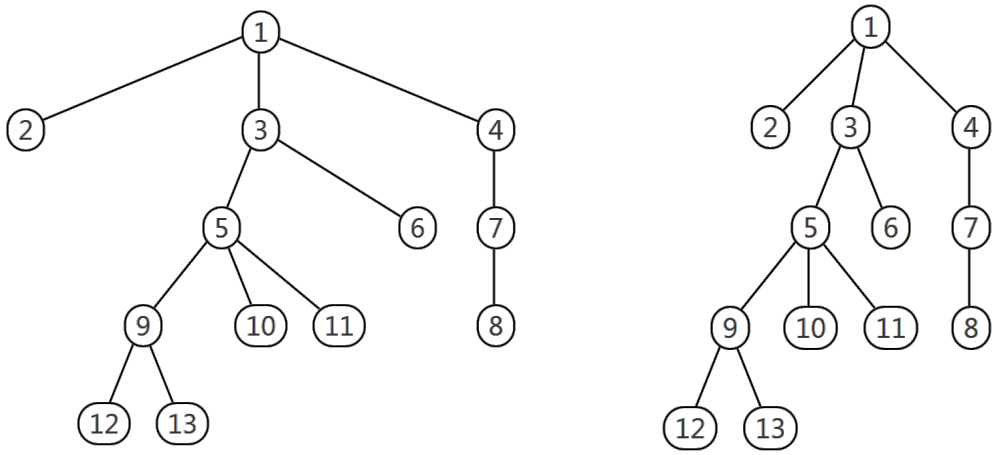
Kapitola 4

Experimenty a výsledky

Program byl napsán v jazyce Java, verze 8. Pro uživatelské rozhraní a vykreslování byl použit framework JavaFX.

4.1 Grafické výsledky

Jak ukazuje Obrázek 4.1b, algoritmus netrpí problémem s prázdnými plochami a jeho výstup má minimální šířku. Zároveň dodržuje hierarchii a pořadí vrcholů. Vykreslení rozsáhlejšího stromu ukazuje Obrázek 4.2.

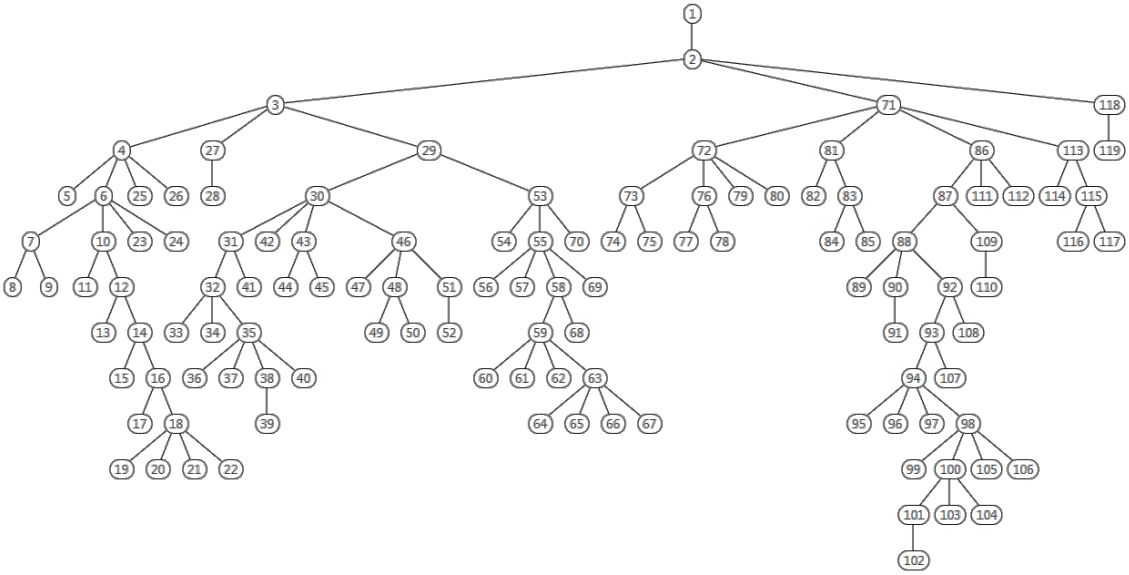


(a) Výstup algoritmu z kapitoly 2.1. (b) Výstup vlastního řešení.

Obrázek 4.1: Porovnání výstupu jednoduchého algoritmu a vlastního řešení.

4.2 Měření rychlosti

Pro měření rychlosti byly vygenerovány náhodné stromy o nastaveném maximálním počtu až 9 následníků a maximální hloubce až 30 úrovní. Měření zahrnovalo pouze běh pozicovacího algoritmu, nikoli samotné vykreslování. Bodle Tabulky 4.1 se dá říci, že rychlost algoritmu určitě nebude při jeho běžném použití překážkou.



Obrázek 4.2: Vykreslení rozsáhlejšího stromu.

Tabulka 4.1: Doby běhů jednotlivých algoritmů na procesoru Intel Core i5-4570. Hodnoty jsou zaokrouhleny na celá čísla, počty vrcholů na tisíce.

Počet vrcholů	Jednoduchý alg.	Vlastní alg.	Vlastní alg. (vylepšený)
3 000	0 ns	1 ns	1 ns
10 000	0 ns	8 ns	7 ns
45 000	2 ns	24 ns	24 ns
125 000	5 ns	65 ns	37 ns
270 000	7 ns	122 ns	77 ns
2 050 000	50 ns	902 ns	408 ns

Kapitola 5

Závěr

Navrhli a implementovali jsme algoritmus pro kreslení obecných n-nárních stromů, který dává dobrý grafický výstup a je dostatečně efektivní na to, aby mohl být použit i v praxi.

V rámci experimentů byla vytvořena jednoduchá aplikace v JavaFX, která dokáže vykreslit stromy nejen navrženým algoritmem popsaném v kapitole 3, ale i jednoduchým rekurzivním algoritmem z kapitoly 2.1. Všechny obrázky použité v této práci od kapitoly 2 byly vykresleny touto aplikací.

Literatura

[CB02] Sebastian Leipert Christoph Buchheim, Michael Jünger. Improving walker's algorithm to run in linear time, 2002.

[MR] Edward a John S. TILFORD M. REINGOLD. Tidier drawings of trees. *IEEE Transactions on software engineering 1981*, SE-7:223–228. URL: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.150.3559&rep=rep1&type=pdf`.

[Ski09] S.S. Skiena. *The Algorithm Design Manual*. Springer London, 2009. URL: `https://books.google.cz/books?id=7XUSn0IKQEgC`.

[Wal90] John Q. Walker. A node-positioning algorithm for general trees. *Software: Practice and Experience*, 20(7):685–705, 1990. URL: `http://dx.doi.org/10.1002/spe.4380200705`, `doi:10.1002/spe.4380200705`.