Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského v Bratislave

Editor zložených grafov

(bakalárska práca)

Autor: Martin Sarvaš

Školiteľ: RNDr. Jana Katreniaková, PhD

9.2.1 Informatika

Bratislava, 2010

Univerzita Komenského v Bratislave
Fakulta Matematiky, Fyziky a Informatiky

EDITOR ZLOŽENÝCH GRAFOV

bakalárska práca

Študijný program:	Informatika
Študijný odbor:	9.2.1 Informatika
Školiace pracovisko:	Katedra Informatiky
Školiteľ:	RNDr. Jana Katreniaková, PhD

Bratislava, 2010

Martin Sarvaš

Čestne prehlasujem, že som bakalársku prácu vypracoval samostatne s použitím uvedenej literatúry.

V prvom rade by som chcel poďakovať Bohu za život a všetko, čo mi dal. Okrem toho moja vďaka patrí mojej školiteľke za odborné vedenie práce a mojej rodine a priateľom, ktorí ma povzbudzovali pri práci.

Abstrakt:
SARVAŠ, Martin: Editor zložených grafov [bakalárska práca]/Martin Sarvaš. -Univerzita Komenského v Bratislave. Fakulta Matematiky, Fyziky a Informatiky; Katedra Informatiky. Školiteľ: RNDr. Jana Katreniaková, PhD. Bratislava FMFI UK, 2010

Cieľom tejto práce je vytvorenie editora pre špeciálny typ grafov nazývaných zložené grafy. Editorom sa dajú vytvoriť, importovať a exportovať do GraphML formátu vlastné zložené grafy s vrcholmi a hranami. Hlavnou výhodou oproti množstvu podobných editorov je možnosť importovať vlastný zobrazovací algoritmus, ktorý upraví rozmiestnenie vrcholov a hrán.

Kľúčove slová: editor, zložený graf, digraf, GraphML, zobrazovací algoritmus

Abstract:
SARVAŠ, Martin: Editor of compound graphs [bachelor's thesis]/Martin Sarvaš. -Komenius Univerzity in Bratislava. Faculty of Mathematics, Physics and Informatics; Department of Computer Science. Advisor: RNDr. Jana Katreniaková, PhD. Bratislava FMFI UK, 2010

The aim of this thesis is to create an editor for a special type of graphs called compound graphs. With editor you can create, import and export to GraphML your own compound graphs, its vertices and edges. The main advantage is possibility to import your own graph visualization algorithm to place the vertices and edges.

Keywords: editor, compound graph, digraph, GraphML, visualization algorithm

Obsah

Kapitola 1

Úvod

V súčastnosti poznáme veľa typov grafov, grafových štruktúr, algoritmov na ich zobrazovanie, prehľadávanie a upravovanie. Práve preto vznikla potreba programov, ktoré by boli otvorené, doplniteľné, upraviteľné na nové typy problémov a ľahko použiteľné na konkrétne problémy. Editor zložených digrafov je jedným z takýchto programov. Zameriava sa na zložené digrafy, čo sú grafy s dvoma typmi hrán. Hierarchické hrany vytvárajú strom a hrany susednosti ďalšie vzťahy. Sú vhodné napríklad na reprezentáciu elektronických kníh a učebných materiálov, ktoré sami o sebe majú hierarchickú štruktúru a zároveň obsahujú hypertextové odkazy na iné časti textu.

V druhej kapitole si stručne definujeme základné pojmy ako graf, sled, cesta, strom, list, potomok, zložený graf a ostatné potrebné termíny. Sú písané jednoduchým štýlom, aby boli pochopiteľné aj pre menej zdatného čitateľa v oblasti grafov. Okrem definícií predstavíme aj zaužívané zobrazovacie štýly pre jednoduché a zložené grafy.

V tretej kapitole sa venujeme formátu GraphML ako štandardu pre grafy. Popíšeme základnú štruktúru ako aj jednoduchý rozširovací mechanizmus. Popis dopĺňame aj jednoduchými príkladmi GraphML.

Štvrtá kapitola popisuje návrh editora, jeho dátové štruktúry pre vrcholy a hrany a algoritmy, ktoré sme použili.

V piatej kapitole bližšie rozoberieme implementáciu, problémy spojené s použitými nástrojmi, podmienky pre správne načítanie grafu z GraphML súboru, vstupný a výstupný formát zobrazovacieho algoritmu, licenciu, pod ktorou vydávame softvér, ako aj špeciálne riešenia, ktoré sa vyskytli pri riešení problémov s implementáciou.

A na koniec v závere zhodnotíme program, či sa nám podarilo naprogramovať to, čo sme chceli.

Kapitola 2

Grafy

2.1 Základné pojmy

V tejto práci budeme uvažovať len o *orientovaných zložených grafoch* alebo *zložených digrafoch*. Skôr ako však definujeme tento špeciálny typ grafov potrebujeme definovať graf ako taký.

Definícia 2.1.1 *Graf je usporiadaná dvojica* $G = (V, E)$*, kde* V *je neprázdna množina všetkých vrcholov grafu* G *a* E *je množina dvojíc z* V *všetkých hrán grafu* G*. Prvky množiny* E *môžu byť usporiadané dvojice z množiny* V $(E \subseteq V \times V)$ *, vtedy hovoríme o orientovaných grafoch alebo digrafoch.*

Definícia 2.1.2 *Pre graf* $G = (V, E)$ *existuje u-v sled* $u = v_0$*,* $v = v_n$ $v_0, l_1, v_1, l_2, v_2, \ldots v_{n-1}, l_n, v_n$*, kde* $v_i \in V$ *sú vrcholy,* $l_i \in E$ *sú hrany ak* $\forall i \in (0..n-1)\,(v_i v_{i+1}) = l_{i+1} \in E$ *medzi dvojicami vrcholov* $(v_i v_{i+1})$ *je hrana. Je to postupnosť vrcholov a hrán.*

Definícia 2.1.3 *Cesta* v_0, v_n *je* v_0*-*v_n *sled, v ktorom sa neopakujú vrcholy ani hrany.* $\forall i, j \in (0..n-1)$*,* $v_i, v_j \in V$ $v_i = v_j \Leftrightarrow i = j \wedge$
$\forall i, j \in (0..n-1)\, v_i v_{i+1}, v_j v_{j+1} \in E$ $v_i v_{i+1} = v_j v_{j+1} \Leftrightarrow i = j$

V teórii grafov sa často stretáme aj s pojmom strom. Pre naše účely budeme uvažovať iba stromy s orientovanými hranami.

Definícia 2.1.4 *Strom je taký graf $T = (V, E)$, v ktorom pre každý vrchol $x \in V$ existuje práve jedna cesta x, y pre $y \in V$.*

Definícia 2.1.5 *List l je taký vrchol stromu $T = (V, E)$, z ktorého nevychádza žiadna hrana. $l \in T$, $\forall v \in T$, $lv \notin E$*

Definícia 2.1.6 *Zakorenený strom T, je taký strom, ktorý má jeden význačný vrchol r_T(koreň) a všetky hrany orientované smerom od r_T k listom.*

V strome môžme uvažovať o špeciálnych funkciách, ktoré zisťujú predkov, potomkov, rodiča a dieťa. Rozoznávajú sa na základe smeru orientovanej hrany. Napr.: Koreň nemá žiadneho rodiča, lebo do neho nevchádzajú žiadne hrany a preto všetky vrcholy s ktorými je spojený hranou sú jeho deti.
Z listu nevychádzajú žiadne hrany, preto nemá potomkov a ak do neho smeruje hrana z iného vrcholu, je to jeho rodič.

Definícia 2.1.7 *Nech $v \in G$, kde $G = (V, E)$, potom:*
$pred\,(v) = \{u | u \in r_T, v\,;\ r_T, v\, je\, cesta\}$, je množina všetkých predkov.
$pot\,(v) = \{u | u \in v, l\,;\ \forall l \in V\,;\ l\, je\, list\, stromu\, T,\, v, l\, je\, cesta\}$, je množina všetkých potomkov
$rodic\,(v) = u$ práve vtedy, keď $\exists\,(u, v) \in E$, je rodič, priamy predok.
$dieta\,(v) = \{u |\,(v, u) \in E\}$, je množina všetkých detí, priamych potomkov.

V jednoduchých grafoch používame hrany na reprezentáciu vzťahu susednosti medzi dvoma vrcholmi. Napríklad vrchol a je incidentný, susedný s vrcholom b, ak je medzi vrcholom a a b hrana V stromoch sa hrany používajú na reprezentovanie vzťahu hierarchie, inklúzie. Zložené grafy sú špeciálnym typom grafu, ktorý spája tieto dva vzťahy susednosti a inklúzie. Obsahuje hrany hierarchického typu aj hrany susednosti.

Definícia 2.1.8 *Zložený graf G je usporiadaná trojica $G = (V, E, F)$, kde $T = (V, E)$ je zakorenený strom T a $P = (V, F)$ je orientovaný graf, kde $\{(u, v) \in F, v \in pot\,(u) \cup pred\,(u)\} = \emptyset$*

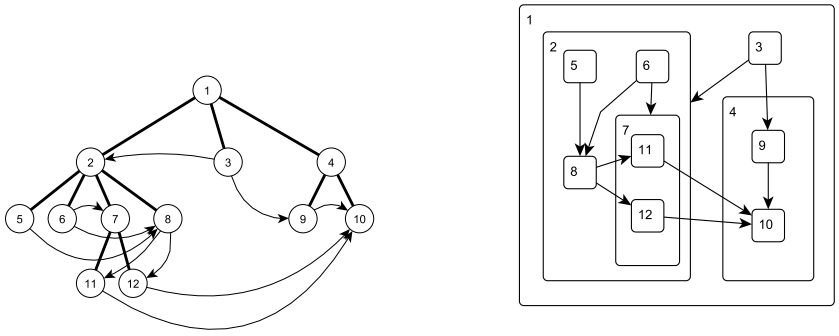
Pre lepšie pochopenie definícií odporúčam prečítať si [1], prípadne [2] alebo [4]

2.2 Zobrazovanie digrafov

Pri vizualizácii sa vrcholy grafu znázorňujú ako kružnice, štvorce alebo obdĺžniky a orientované hrany ako lomené šípky. Stromy zobrazujeme tak, že koreň vykreslíme ako prvý a všetky ostatné vrcholy nižšie, pričom ich zoraďujeme podľa úrovní tak, že do nasledujúcej úrovne zobrazíme len tie vrcholy, čo sú spojené hranou s vrcholmi z aktuálnej úrovne.

2.2.1 Zobrazovanie zložených digrafov

Pri zložených grafoch použitím analógie dospejeme ku verzii nakresliť si strom $T = (V, E)$ s hierarchickými hranami, bez šípok, lebo tam je orientácia jasná, a potom prikresliť hrany susednosti $P = (V, F)$ so šípkami ako na obrázku 2.1(a).



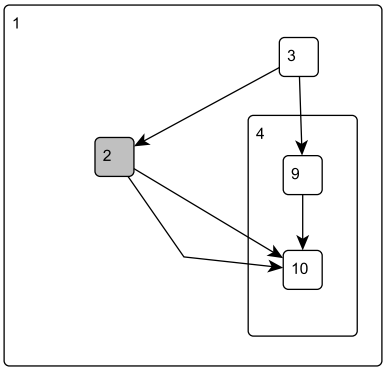
(a) (b)

Obr. 2.1: Zložený graf

Toto riešenie nie je úplne zlé, ale pri väčších a zložitejších grafoch neprichádza v úvahu. Lepšie riešenie je reprezentovať hierarchické hrany iným spôsobom ako čiarou. A keďže je to vzťah inklúzie, pomôžeme si zobrazovaním množín. Budeme teda hierarchické vzťahy reprezentovať vnorením, geometrickou inklúziou. Keď má teda koreň všetky vrcholy podradené, budú

vpísané do neho ako na obrázku 2.1(b). Hrany susednosti budeme kresliť naďalej šípkami.

Pri zobrazovaní zložených grafov môžeme vrchol nazvať aj *cluster*. Clustre môžu byť kontrahované alebo expandované. *Expandované clustre* však môžu byť len vrcholy, čo majú deti, teda nie sú listami a všetky jeho deti sú vykreslené ako na obrázku 2.2 vrchol 2. *Kontrahovaný cluster* je buď list alebo vrchol, ktorý v sebe obsahuje ďalšie vrcholy a hrany, no nevykreslia sa. Vykresľuje sa iba cluster ako jednoduchý vrchol, s tým, že všetky hrany, čo smerovali do neho a do jeho potomkov, budú smerovať priamo do clustra a opačne ako na obrázku 2.2 vrchol 2, 3. Podrobnejšie v [2].



Obr. 2.2: Zložený graf s kontrahovným clustrom

Kapitola 3

Reprezentácia grafov

Jednou z možností ako reprezentovať graf v elektronickej podobe je formát GraphML. Je to otvorený a ľahko rozšíriteľný formát a práve preto sme si ho vybrali. GraphML pozostáva z jadra jazyka, ktorým sa popisuje štruktúra a flexibilným rozšíriteľným mechanizmom na popísanie špecifických dát [5]. Formát GraphML je založený na XML formáte a tým získal výhodu oproti ostatným formátom reprezentujúcimi grafy. Dá sa preto jednoducho používať a existuje veľa podporných programov na spracovanie XML. Licencia na používanie formátu je „Creative Commons Attribution 3.0 License“, čo znamená že môže byť použitý bez poplatku pre vedecký aj komerčný softvér.

3.1 Popis štruktúry v GraphML

GraphML sa ako každé XML začína elementom `<?xlm>`. Prvým párovým tagom je `<graphml> </graphml>`, v ktorom sa nachádza celý graf. Potom môže nasledovať tag `<key>`, v ktorom sa dajú definovať vlastné atribúty. Samotný graf je vnorený ešte do jedného párového tagu `<graph> </graph>`. Ten obsahuje elementy `<node> </node>`, ktoré reprezentujú vrcholy a všetky elementy `<edge> </edge>` reprezentujúce hrany. V elemente `<node>` sa môže nachádzať aj element `<graph>`, pomocou ktorého môžeme vytvoriť zložené grafy. V elemente `<graph>` je povinný atribút `edgedefault`, ktorým sa dá

naspaviť, či bude graf s orientovanými, neorientovanými alebo zmiešanými hranami. Pre naše účely bude stále `edgedefault=directed`, čiže s orientovanými hranami. Element `<edge>` má povinné dva atribúty pre začiatok a koniec hrany `source` a `target`. Pre element `<node>` je povinný atribút len atribút `id`.

Príklad:

```
<?xml version="1.0" encoding="UTF-8"?><graphml
    xmlns="http://graphml.graphdrawing.org/xmlns"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
    http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd" >
  <graph id="G" edgedefault="directed">
    <node id="n0"/>
    <node id="n1"/>
    <node id="n2"/>
    <node id="n3"/>
    <node id="n4">
        <graph id="n4:" edgedefault="directed">
          <node id="n4::n0"/>
          <node id="n4::n1"/>
          <node id="n4::n2"/>
          <edge source="n4::n0" target="n4::n2"/>
          <edge source="n4::n1" target="n4::n2"/>
        </graph>
    </node>
    <edge source="n0" target="n2"/>
    <edge source="n1" target="n2"/>
    <edge source="n2" target="n3"/>
    <edge source="n3" target="n4::n1"/>
  </graph>
</graphml>
```

3.2 Rozšíriteľnosť v GraphML

Formát GraphML nemá veľa základných atribútov, ktoré by sme mohli použiť na uchovávanie dodatočných informácií o grafe, vrcholoch a hranách. Práve preto v ňom existujú dva mechanizmy na rozšírenie základnej štruktúry. Jedným je už spomínaný tag `<key>` a druhým je možnosť definovať si vlastné tagy. Druhú možnosť však pre naše účely nebudeme potrebovať. Tag `<key>` má jeden povinný atribút `id`, ale pravdepodobne budeme využívať aj atribút `for, attr.name` a `attr.type`. S tagom `<key>` dokážeme definovať novú vlastnosť pre vrchol, hranu alebo graf pomocou atribútu `for`. Nazveme novú vlastnosť a definujeme typ vlastnosti pomocou `attr.name` a `attr.type`. Potom v jednotlivých elementoch vrchola, grafu alebo hrany použijeme tag `<data>` s atribútom `key`, ktorému nastavíme príslušné `id` vlastnosti.

Príklad:

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"  ... >
  <key id="d0" for="node" attr.name="color" attr.type="string"/>
  <key id="d1" for="edge" attr.name="weight" attr.type="double"/>
  <graph id="G" edgedefault="directed">
    <node id="n0">
      <data key="d0">green</data>
    </node>
    <node id="n1">
      <data key="d0">blue</data>
    </node>
    <edge id="e0" source="n0" target="n1">
      <data key="d1">1.0</data>
    </edge>
  </graph>
</graphml>
```

Kapitola 4

Návrh implementácie

Našou úlohou bolo naprogramovať editor, v ktorom by sa dali vytvárať, zobrazovať a upravovať zložené digrafy a dal sa importovať externý algoritmus na určenie polohy jednotlivých vrcholov. Preto prioritou bola základná funkčnosť a ako doplňujúce faktory boli otvorenosť a ľahká použiteľnosť.

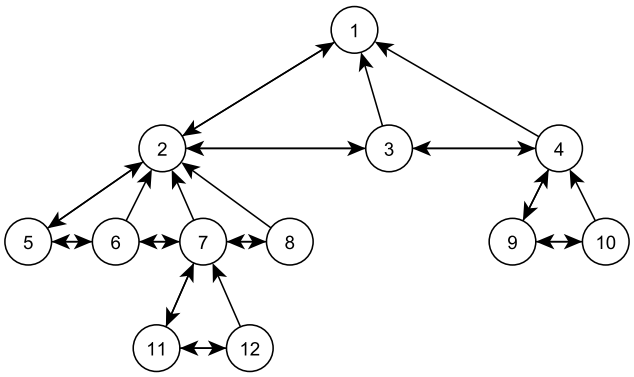
4.1 Návrh dátového modelu

Najdôležitejšou časťou návrhu dátového modelu je výber internej štruktúry, v ktorej budeme uchovávať informácie o vrcholoch a hranách a výber algoritmov na prácu s dátami.

4.1.1 Návrh štruktúry a algoritmov pre vrcholy

V návrhu dátového modelu hlavnú úlohu zohral rýchly a jednoduchý export a import grafu do formátu GraphML. Hierarchická štruktúra zloženého grafu je strom a pri prehľadávaní do hĺbky prechádzame vrcholy v poradí ako vo formáte GraphML. Teda v snahe čo najviac optimalizovať operáciu na export do GraphML, vloženie vrchola, nájdenie vrchola a zmazanie vrchola sme vymysleli novú dátovú štruktúru. Je to štruktúra podobná zakorenenému orientovanému stromu, v ktorej má vrchol smerník iba na otca, prvého syna

a pravého a ľavého brata ako na obrázku 4.1.1. Pracovne sme si ju nazvali *FS2B*(Father, Son, 2 Brothers). Dajú sa nad ňou vykonávať operácie vloženie vrchola, vymazanie vrchola, nájdenie vrchola, úprava vrchola a ďalšie.

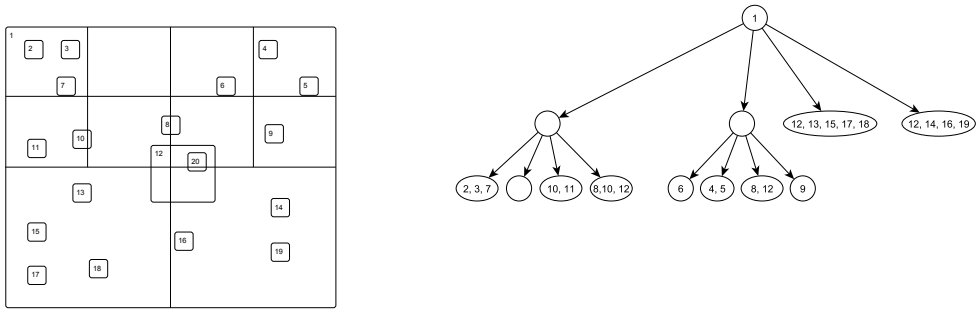


Obr. 4.1: Štruktúra pre vrcholy FS2B

Algoritmus na nájdenie vrchola funguje tak, že po zadaní počiatočného vrchola a súradníc x, y porovná súradnice vrchola, či sa v ňom nenachádza konkrétna súradnica, ak áno rekurzívne sa zavolá tá istá metóda s prvým synom, ak nie zavolá sa so všetkými ostatnými synmi resp. pravými bratmi prvého syna. Ak prejdeme všetkých pravých bratov, tak vráti otca. Čitateľ s minimálnymi znalosťami z oblasti zložitosti a dátových štruktúr ľahko nahliadne, že zložitosť tohto algoritmu bude v najhoršom prípade $O(n)$, kde n je počet vrcholov.

Pri grafoch, čo majú na jednej úrovni väčšinu vrcholov, by to mohol byť problém. Preto by bolo vhodné, pri vrcholoch, čo majú nad istý počet vrcholov usporiadať ich do štruktúry *quadtree* [6]. Quadtree je stromová štruktúra, v ktorej má vrchol vždy 4 synov a zvyčajne sa používa na delenie dvoj-

dimenzionálneho priestoru. Funguje podobne ako binárny vyhľadávací strom so 4 synmi. Každý syn obsahuje bunku, ktorá reprezentuje istú časť priestoru vrchola. Ak počet synov daného vrchola prekročí istú hranicu tak sa bunka rozdelý na 4 rovnaké časti a synovia daného vrchola sa tam premiestnia, podľa toho, kde ktorý patrí. Ak by vrchol zasahoval do viacerých buniek quadtree, tak sa vloží do každej bunky do ktorej spadá ako na obrázku 4.2.



(a) (b)

Obr. 4.2: Quadtree

4.1.2 Návrh štruktúry a algoritmov pre hrany

Hrany susednosti potrebujeme vedieť exportovať do GraphML, zobrazovať, meniť v závislosti od zmeny vrcholov, označovať, upravovať a mazať. Vhodnou štruktúrou, ktorou by sa dali uchovať hrany, môže byť aj jednoduchý *map* obsahujúci zloženú štruktúru. Map je usporiadané asociatívne pole[7], do ktorého sa dajú priradiť prvky *mapped value* na základe tzv. *key value*. Pričom mapped value môže byť prvok takmer akéhokoľvek typu, v ktorom sú uložené dáta. V našom prípade by to bola zložená štruktúra, ktorá by mala dva smerníky na vrcholy, ktoré spája, rovnicu priamky a ďalšie informácie

o hrane, ktoré si potrebujeme zapamätať. Operácie zobrazenie, úprava, mazanie a export hrany do GraphML potrebujú ako vstup identifikátor hrany. A keďže máme indexované hrany v štruktúre map pomocou identifikátorov hrán a usporiadané asociatívne pole má garantovanú zložitosť $O\left(log\left(n\right)\right)$ [8], mali by mať aj tieto operácie zložitosť $O\left(log\left(n\right)\right)$, kde n je počet hrán.

Na to aby sme vedeli efektívne meniť hrany v závislosti od zmeny vrcholov, potrebujeme ešte štruktúru v ktorej si budeme pamätať aké hrany prislúchajú ku konkrétnemu vrcholu. Potom pri zmene vrchola zaktualizujeme iba tie hrany, ktorých sa to týka. Jednoduché riešenie je zapísať tieto informácie do už existujúcej štruktúry pre vrcholy FS2B. Stačí vytvoriť niečo ako spájaný zoznam v každom vrchole, v ktorom si budeme pamätať jednotlivé identifikátory hrán.

Na označovnie hrán potrebujeme smernicovú rovnicu priamky $y = k * x + q$, kde k je tangens uhla, ktorý zviera x-ová súradnica s priamkou a q je posunutie, o koľko je posunutá priamka na osi y z bodu $[0, 0]$. Na zistenie, či daný bod $p\left[x, y\right]$ leží na priamke stačí dosadiť x do rovnice priamky a porovnať, či sa rovná y. V realite ale musíme kontrolovať aj úzke okolie priamky na istom úseku $\langle x_1, x_2\rangle$ a $\langle y_1, y_2\rangle$. So smernicovou rovnicou priamky sa to dá jednoducho posunutím q o požadovanú vzdialenosť d ($y < k * x + q + d$; $y > k * x + q - d$) a porovnávaním bodov v prieniku polrovín, ktoré vyčleňujú tieto dve priamky. Pri určitom počte hrán je zbytočné optimalizovanie označovnia hrán, preto stačí skontrolovať postupne všetky hrany. Zložitosť algoritmu bude teda $O\left(n\right)$, kde n je počet hrán.

Pre efektívnejšie označovanie vrcholov by sa dala použiť pomocná štruktúra *centered interval tree*[9]. V reálnom použití, by však bolo toto vylepšenie nepatrné, pretože priestor a čas využitý na udržovanie štruktúry pri malých grafoch je podstatne väčší ako úspora. Pri obrovských grafoch, kde je naozaj veľa hrán to však môže zohrať významnú úlohu pri optimalizácii.

Základnú funkcionalitu, teda možnosť importovať vlastný zobrazovací algoritmus dosiahneme tak, že celý graf exportujeme do GraphML súboru, ten prepíšeme algoritmom a následne importujeme naspäť do editora. Na export a import GraphML je dobré použiť už existujúci parser XML, pretože GraphML je XML formát, XML je dosť rozšírený a existuje preň množstvo riešení.

Kapitola 5

Imlementačné detaily

Pri implementácii kľúčovu úlohu tvorí správny výber nástrojov, platformy a programovacieho jazyka. Veľa ráz práve nesprávny výber nástrojov dokáže ohroziť softvérový projekt tak, že skončí neúspechom. V tejto kapitole preto predstavíme vybrané nástroje, definujeme vstup a výstup pre GraphML súbory a predstavíme formát vstupu a výstupu pre zobrazovacie algoritmy.

5.1 Výber nástrojov na implementáciu

Prvou možnosťou pri implementácii bolo rozšírenie voľne dostupného editora grafov na požadovanú funkčnosť. Z množstva riešení, ktoré som skúmal by som rád spomenul Graphviz, otvorený program na vizualizáciu grafov [10]. Je to silný nástroj, má však jednu nevýhodu. Je založený na jazyku DOT, čo je jazyk na popis grafov. Pre tento projekt, založený a optimalizovaný pre formát GraphML, je však nevhodný. Ďalším zaujímavým programom je uDraw(Graph) [11]. Rovnako zaujímavý nástroj na tvorbu grafov, avšak nedajú sa v ňom umiestňovať vrcholy na ľubovoľnú pozíciu. Najlepším a najvhodnejším editorom sa zdal byť yEd Graph Editor [12], ktorý je založený na XML a formáte GraphML. Žiaľ zatiaľ uzavretý softvér s nepoužiteľnou licenciou pre náš projekt. Ostatné riešenia, ak mali dobrú licenciu používali nevhodný jazyk alebo zlý formát. Preto som sa rozhodol, že pre tento projekt

bude lepšie implementovať ho celý od začiatku.

Pri vlastnej implementácii je dôležité vybrať si vhodný multiplatformový jazyk a nástroje. Rozhodol som sa pre jazyk C++ na odporúčanie mojej vedúcej a vzhľadom na jeho multiplatformovosť. Používal som grafickú knižnicu wxWidgets 2.8.9 [13] a k tomu grafické prostredie wxDev-C++ [14]. Knižnica wxWidgets je síce multiplatformová, avšak nie je až tak známa ako knižnica Qt. Páčila sa mi však natívna podpora wxDev-C++, preto som sa rozhodol použiť túto kombináciu. Počas práce som však toto riskantné rozhodnutie oľutoval, pretože mi prestalo fungovať debuggovanie vo wxDev-C++ a tým spomalilo vývoj editora. A keďže už bolo neskoro pretransformovať celý projekt, musel som ho dokončiť bez podpory debuggovania.

GraphML je XML formát, preto na načítanie a uloženie grafu do GraphML som použil XML parser TinyXML[16]. Rozhodoval som sa ešte medzi parserom Xerces-C++ [18] a parserom CodeSynthesis XSD[19]. Xerces-C++ je parser aj na validáciu XML súborov, preto je dosť objemný a pre Windows platformu je skompilovaný iba pre MS Visual C++. Všetky moje snahy sfukčniť Xerces-C++ skončili neúspechom. CodeSynthesis XSD, ako som neskôr zistil, používa Xerces-C++ a preto som ho nepoužil, hoci sľuboval menej programovania ako TinyXML.

5.2 Načítanie grafu z Graphml súboru

Editor by nemal mať problém načítať základnú štruktúru grafu, teda vrcholy a hrany medzi nimi, z akéhokoľvek validného GraphML súboru. Pre správne umiestnenie vrcholov musia byť splnené nasledujúce kritériá:

- správne definované `<key>` elementy pre atribúty vrchola `x`,`y` pre umiestnenie, `width` pre šírku a `height` pre výšku vrchola
- použité prislúchajúce idetifikátory k jednotlivým `<key>` elementom v elementoch `<data>` vo vrcholoch

V prípade, že nie sú zadané súradnice vrchola alebo sú zle zadané, editor ich vykreslí do stredu. Ak je zle zadaná výška alebo šírka editor priradí daným vrcholom štandardnú výšku resp. šírku. V teórii zložených digrafov je definované, že všetky vrcholy musia byť potomkami jedného koreňového vrchola. Tento koreňový vrchol berieme pri načítavaní ako prvý element `<graph>`, ktorý môže mať definovanú šírku aj výšku podobne ako element `<node>`. Pri načítaní grafu zo súboru a opätovnom uložení editor zatiaľ nezachováva pôvodný súbor ani žiadne informácie, ktoré nie sú načítané.

Príklad načítateľného vstupu:

```
<?xml version="1.0" encoding="UTF-8" ?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
    <key id="d0" for="node" attr.name="x" attr.type="int"/>
    <key id="d1" for="node" attr.name="y" attr.type="int"/>
    <key id="d2" for="all" attr.name="width" attr.type="int"/>
    <key id="d3" for="all" attr.name="height" attr.type="int"/>
    <graph id="G" edgedefault="directed">
        <node id="n0">
            <data key="d0">345</data>
            <data key="d1">102</data>
            <data key="d2">30</data>
            <data key="d3">24</data>
        </node>
        <node id="n1">
            <data key="d0">229</data>
            <data key="d1">106</data>
            <data key="d2">50</data>
            <data key="d3">60</data>
        </node>
        <edge id="e0" source="n0" target="n1"/>
    </graph>
</graphml>
```

Pre jednoduchosť dokáže editor importovať aj XML súbor, ktorý je well-formed GraphML avšak obsahuje vo všetkých elementoch `<node>` atribúty `X,Y` pre polohu a `Width, Height` pre šírku a výšku vrchola ako je nižšie popísaný výstup zobrazovacieho algoritmu. Vtedy budú elementy `<data>` vo vrcholoch ignorované.

5.3 Formát vstupu a výstupu zobrazovacieho algoritmu

Hlavnou výhodou tohto editora oproti ostatným grafovým editorom je možnosť zobraziť graf pomocou vlastného zobrazovacieho algoritmu. Keďže táto práca nadväzuje na prácu môjho kolegu [3], ktorý implementoval jeden zobrazovací algoritmus, preto aj v tejto práci použijeme rovnaký vstup aj výstup.
Vstup definoval ako korektný GraphML súbor. V editore je to implementované tak, že sa graf uloží celý do dočasného súboru a algoritmu sa pošle tento súbor ako command-line parameter.

Výstup definoval ako korektný XML súbor s tým, že do dát pôvodného súboru do elementu `<node>` sa pridajú atribúty `X,Y` pre polohu a `Width, Height` pre šírku a výšku vrchola a pošlú sa na štandardný výstup.

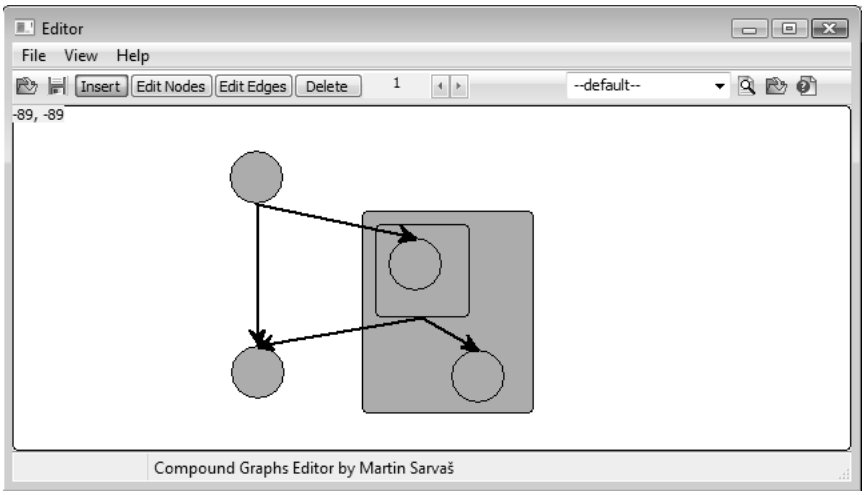
Editor však okrem tohto vstupu dokáže akceptovať aj dáta pôvodného súboru so zmenenými atribútmi v `<data>` elementoch vo vrcholoch, pretože používa tú istú metódu ako na načítanie GraphML súboru.

5.4 Možnosti editora

V návrhu sme spomínali, že medzi základnú funkčnosť okrem importu vlastného algoritmu je možnosť vytvárať, zobrazovať a upravovať zložené digrafy. Preto si teraz predstavíme, čo všetko sa nám podarilo implementovať. Vnútorné štruktúry sú implementované ako v návrhu, až na quadtree, ktorý sa nepodarilo implementovať pre časovú tieseň a nepotrebnosť tak silnej optimalizácie.

5.4.1 Editácia grafu

Editor má základné 3 módy: Vkladací(Insert), úpravu vrcholov(Edit Nodes) a úpravu hrán(Edit Edges). Vo vkladacom móde sa klikutím ľavým tlačidlom na bod vytvorí nový vrchol. Pri prvom kliknutí sa však vytvorí iba koreňový vrchol, ktorý dostane veľkosť aktuálnej obrazovky. Ak klikneme do už existujúceho vrchola, pridáme jeho syna. Ak je pôvodný vrchol malý, zväčší sa, aby sa nový vrchol do neho zmestil ako na obrázku 5.1. Ak stlačíme ľavé



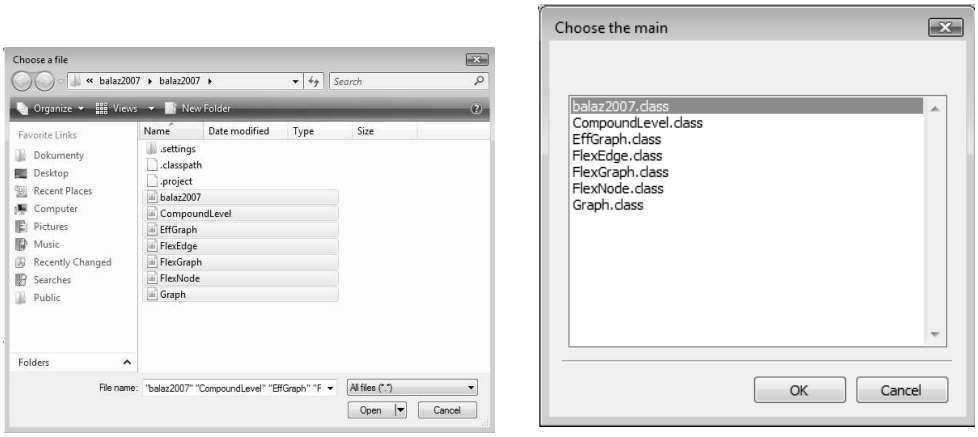
Obr. 5.1: Vkladanie vrcholov a hrán

tlačidlo na jednom vrchole a pustíme na druhom vytvorí sa hrana, ak nie je

porušené kritérium z teórie o zložených digrafoch(hrana nemôže spájať dva vrcholy, z ktorých jeden je predok druhého).

V móde na úpravu vrcholov sa potom dajú kliknutím označovať jednotlivé vrcholy. Pri stlačení a ťahu myšou sa dajú premiestniť. Ak stlačíme tlačidlo „delete“ a máme označený vrchol, tak ho týmto činom vymažeme aj so všetkými jeho potomkami a hranami, s ktorými je incidentný on ako aj jeho potomkovia.

V móde na úpravu hrán môžme kliknutím na čiaru hrany ozanačiť hranu. Podobne ako u vrcholou kliknutím na tlačidlo delete vymažeme označenú hranu.



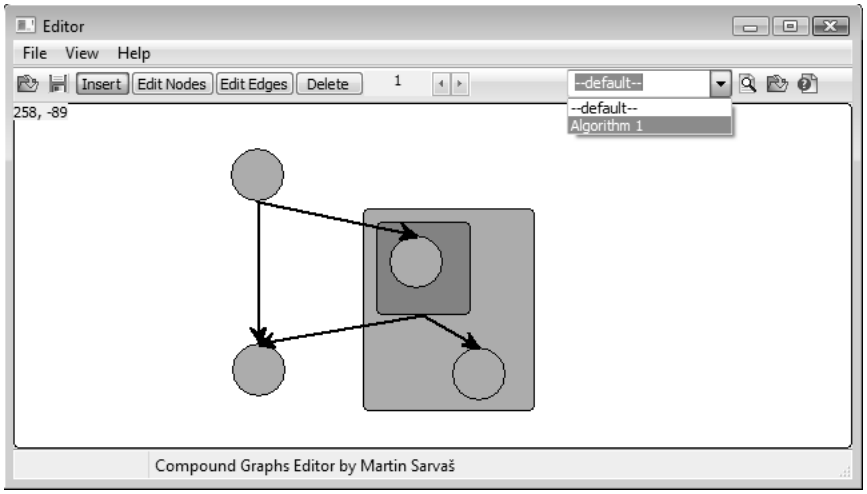
(a) (b)

Obr. 5.2: Pridanie algoritmu

5.4.2 Zobrazovací algoritmus

Pri pridávaní nového zobrazovacieho algoritmu treba označiť všetky súbory, ktoré patria k algoritmu ako na obrázku 5.2. Ak ich bolo viac, tak potom treba vybrať, ktorý zo súborov obsahuje main funkciu. Editor dokáže spustiť súbory s príponou *.class a *.exe.

Nakoniec ešte treba pridať názov, pod ktorým ho chceme mať identifikovaný. Po pridaní algoritmu ho môžme použiť. Zmeníme v listovom menu hodnotu „–default–“ na názov algoritmu a stlačíme príslušné tlačidlo na zobrazenie(View graph), obr. 5.3.



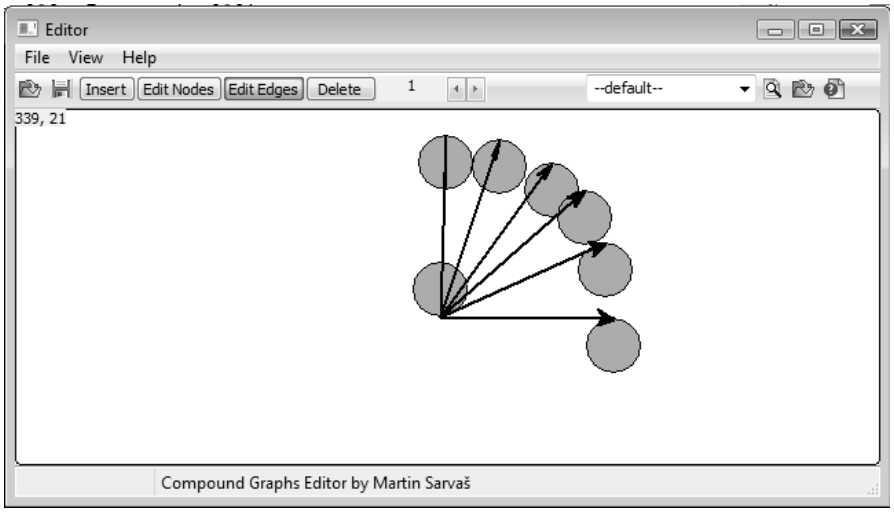
Obr. 5.3: Výber algoritmu

5.5 Licencovanie

Parser TinyXML je vydaný pod licenciou zlib [17]. A keďže tá je kompatibilná s GPLv3, výsledný editor vydávam pod licenciou GNU General Public License (GPL) version 3 [15].

5.6 Špeciálne riešenia

Medzi zaujímavosti implementácie patrí určite vykresľovanie hrán. Pokiaľ by sme mali neorientované grafy, tak spojiť dva body je triviálne. Problém nastáva, keď máme nakresliť šípku. V princípe by to nemal byť problém, stačí len na koniec úsečky vykresliť v správnom smere vyfarbený polygón. A aby bola šípka v správnom smere si predsa vieme vypočítať pomocou smernicovej rovnice priamky $y = k*x+q$. Celkom jednoduché riešenie, problém ale začína, keď chceme mať šípku umiestnenú presne vertikálne, kde k z rovnice priamky nadobúda hodnotu $\pm\infty$. Pre riešenie tohto zaujímavého problému sme sa



Obr. 5.4: Problémové kreslenie šípky

rozhodli opäť použiť matematiku. Zadelíme si teda smernice priamok do 2 oblastí.

1. $k = (0, 1) \cup (-1, 0)$
2. $k = (1, \infty) \cup (-\infty, -1)$

V prvej oblasti nemáme absolútne žiaden problém pri vykresľovaní na rozdiel od druhej, kde pri veľkých číslach k sa šípky zužujú, až na koniec zmiznú keď $k = \infty$ ako na obrázku 5.4. Riešením by mohlo byť zobrazenie bodov a priamok do oblasti 1., kde sa vykoná výpočet súradníc bodov a spätne sa zobrazia do oblasti 2. Inverzné zobrazenie spĺňa všetky požadované kritériá a je pritom veľmi jednoducho a elegantne implementovateľné, stačí vymeniť hodnoty x a y.

Kapitola 6

Záver

Podarilo sa nám implementovať veľa z pôvodného návrhu editora. Spĺňa základné kritéria, ktoré sme si definovali na začiatku. Editorom sa dajú vytvárať, upravovať a zobrazovať zložené digrafy. A keďže sme softvér vydali pod licenciou GPLv3, splnili sme aj doplňujúcu požiadavku na otvorenosť. Poslednú požiadavku, ľahkú použiteľnosť, však musia otestovať a ohodnotiť používatelia. Pri programoch tohto typu je vždy čo vylepšovať a veľa sa zmení po nasadení do reality vzhľadom na požiadavky používateľov. Verím, že editor zložených grafov bude úspešným softvérom a ešte o ňom budeme počuť.

Dodatok A

Príloha

K práci je priložené CD-médium, na ktorom sa nachádza zdrojový kód editora ako aj skompilovaný spustiteľný súbor pre platformu Windows.

Literatúra

[1] Sugiyama and Misue. 1991. Visualization of structural information: Automatic drawing of compound digraphs. IEEE Trans. on Systems, Man and Cybernetics, 1991.

[2] Katreniaková J., Vizualizácia učiva v elektronickom vzdelávaní pomocou kreslenia zložených grafov, katreniakova@dcs.fmph.uniba.sk

[3] Baláž M. 2007. Algoritmy pre vizualizáciu zložených grafov: bakalárska práca. Braislava: Univerzita Komenského, 2007. 31 s.

[4] Wikipédia. Definícia grafov. http://cs.wikipedia.org/wiki/Graf_%28teorie_graf%C5%AF%29 (19.5.2010)

[5] GraphML. http://graphml.graphdrawing.org/ (21.5.2010)

[6] Wikipédia. Definícia štruktúry Quadtree. http://en.wikipedia.org/wiki/Quadtree(24.5.2010)

[7] Wikipédia. Definícia štruktúry Map. http://en.wikipedia.org/wiki/Map_%28C%2B%2B%29 (2.6.2010)

[8] SGI, Standard Template Library, C++. Sorted Associative Container. http://www.sgi.com/tech/stl/SortedAssociativeContainer.html (2.6.2010)

[9] Wikipédia. Definícia štruktúry Interval Tree.http://en.wikipedia.org/wiki/Interval_tree(3.6.2010)

[10] Graphviz. Softvér. http://www.graphviz.org/ (9.6.2010)

[11] uDraw(Graph). Softvér. http://www.informatik.uni-bremen.de/uDrawGraph/en/home.html (9.6.2010)

[12] yEd Graph Editor. Softvér. http://www.yworks.com/en/products_yed_about.html (9.6.2010)

[13] wxWidgets. Knižnica. http://docs.wxwidgets.org/2.8.9/ (9.6.2010)

[14] wxDevC++. Softvér. http://wxdsgn.sourceforge.net/ (9.6.2010)

[15] GPLv3. Text licencie. http://www.gnu.org/licenses/gpl.html (9.6.2010)

[16] TinyXML. Softvér.http://www.grinninglizard.com/tinyxml/ (9.6.2010)

[17] zlib. Text licencie. http://www.gzip.org/zlib/zlib_license.html (9.6.2010)

[18] Xerces-C++. Softvér. http://xerces.apache.org/xerces-c/index.html (9.6.2010)

[19] CodeSynthesis XSD. Softvér. http://www.codesynthesis.com/products/xsd/ (9.6.2010)