PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA MATEMATICKÉ INFORMATIKY

DIPLOMOVÁ PRÁCE

Algoritmy pro kreslení uspořádaných množin a svazů

2003 Jan Outrata

Katedra matematické informatiky PřF UP Olomouc

Předpis pro vypracování diplomové práce

I.

Při zpracování diplomové práce student usiluje o co nejvýstižnější a nejpříznivější obraz o svých schopnostech, úrovni svých znalostí a osvědčuje, jak si osvojil nezbytné návyky odborného a technického způsobu vyjadřování, jaké má znalosti odborné literatury a jak jí umí používat.

Diplomová práce se v tomto smyslu hodnotí jako celek.

II.

Student se musí ve své práci bez újmy na úplnosti vyjadřovat stručně, odborně, slohově i gramaticky správně. Text (včetně popisků příloh) diplomové práce musí být před odevzdáním pečlivě prohlédnut. Písařské chyby, chyby v tisku apod., musí být opraveny.

Nedostatky diplomové práce v tomto směru snižují klasifikaci i práce jinak obsahově dobré.

III.

Text diplomové práce musí být zpracován programem $\LaTeX$ s použitím makra katedry matematické informatiky a vytištěn po jedné straně papíru formátu A4. Diplomová práce se odevzdává takto:

- 1x originál v knihařské vazbě
- 1x kopie spojená v polotuhých deskách
- 1x záznam textu diplomové práce pořízený textovým editorem na disketě, kterou si student před odevzdáním vyzvedne na sekretariátě katedry.

IV.

V originále se na stránku s místopřísežným prohlášením připojí vlastnoruční podpis.

Došlo-li v průběhu zpracování diplomové práce k významným odchylkám od zadání diplomového úkolu, které na základě žádosti studenta schválil vedoucí katedry, připojí se další list, na němž budou tyto skutečnosti uvedeny. Způsob uvedení se stanoví individuálně a v originále bude k záznamu připojen podpis vedoucího katedry. Nevýznamné odchylky uvádí student v anotaci.

V.

Uspořádání odevzdaných vyhotovení diplomové práce se předepisuje následovně:

- předsádkový list
- zadání diplomové práce
- předpis pro vypracování diplomové práce
- místopřísežné prohlášení o samostatném vypracování diplomové práce
- (list s vyjádřením k odchylkám od zadání, pokud byly schváleny)
- anotace diplomové práce
- obsah diplomové práce s odkazy na odpovídající stránky nebo čísla příloh, seznam tabulek a obrázků
- rozvedení diplomového úkolu podle obsahu s dílčími závěry na konci každé kapitoly
- celkový závěr diplomové práce
- cizojazyčné resumé (cca 15 řádků)
- seznam použité literatury v abecedním autorském uspořádání – tabulky, nákresy, přílohy (doklady, prospekty, elaboráty apod.)

Přední deska originálu bude potištěna stejně jako předsádkový list, je možno vypustit znak univerzity.

Diplomová práce, která nebude vyhovovat tomuto uspořádání, nemůže být přijata.

VI.

Použitá literatura, předlohy apod., použité při zpracování musí být na příslušných místech v diplomové práci označeny odkazem na průběžné číslo ze seznamu použité literatury, příslušná stránka se uvede v hranaté závorce. Jde-li o citát, uvedou se čísla prvního a posledního řádku citátu.

Součástí řešení diplomového úkolu je zdrojový text programu. Způsobilost provozu produktu osvědčuje student při obhajobě diplomové práce.

VII.

Všechny propočty nebo výpočty musí být podrobně a přehledně uspořádány tak, aby každý odborník byl schopen jejich správnost přezkoušet. U použitých vzorců, součinitelů nebo hodnot z praxe musí být uveden původ. Jsou-li uváděny údaje, které mohou tvořit předmět hospodářského nebo státního tajemství, je třeba tuto okolnost uvést při odevzdání diplomové práce. V takovém případě se pro přístup k diplomové práci předepíše zvláštní režim (závazný i pro oponenty).

V Olomouci dne 13. března 1996

Doc. RNDr. Josef Hos
vedoucí katedry
matematické informatiky

Za správnost: Zd. Nesvadbová

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval samostatně.

3. dubna 2003 Jan Outrata

Anotace

Cílem této práce bylo prozkoumat, implementovat a porovnat různé metody automatického generování diagramu uspořádaných množin, speciálně svazů. Vstupem je pouze výčet prvků uspořádané množiny a vlastní relace uspořádání nebo ručně vytvořený prvotní návrh diagramu, výstupem pak co možná nejčitelnější diagram, který zároveň splňuje určité konvence a požadavky. Postupy generování diagramu zahrnují známé i méně známé metody generování diagramů a grafů, ale také jednu vlastní metodu. Vygenerovaný diagram lze dále ručně upravovat a vylepšovat. Nakonec jej lze exportovat do METAPOSTu nebo Zapouzdřeného PostScriptu, nativním formátem pro uložení množiny a diagramu je XML. Aplikace byla napsána primárně pro unixové systémy, ale je portována také na systémy Windows. Aplikace je plně internacionalizována, aktuálně existují anglický a český překlad.

Děkuji Doc. RNDr. Radimu Bělohlávkovi, Dr., Ph.D., za odborné vedení mé diplomové práce, poskytnutí těžko dostupné odborné literatury a také za umožnění volnosti při realizaci. Zároveň bych chtěl na tomto místě poděkovat Mgr. Vilému Vychodilovi a Bc. Miroslavu Kuře za rady, podnětné připomínky a nápady. Rodičům děkuji za vytvoření vhodých pracovních podmínek.

Obsah

Seznam obrázků

Seznam tabulek

1. Úvod

Vizualizace konceptuálních struktur je klíčová a potřebná v mnoha odvětvích vědy a jejích aplikacích. Graf je abstraktní struktura, která se používá pro modelování informací. Reprezentuje informace modelované jako objekty a propojení mezi těmito objekty. Tedy spousta systémů vizualizace informací potřebuje kreslit grafy, které jsou snadno čitelné a srozumitelné. Kreslení grafů adresuje problém konstrukce geometrické reprezentace těchto grafů a podobných struktur. Vizualizace obecně příbuzenských struktur je užitečná v tom, že graf či diagram efektivně vyjadřuje informaci obsaženou ve struktuře. Dobrý diagram pomáhá porozumět systému, špatný může být naopak matoucí a zavádějící.

Automatické generování zobrazení grafů a diagramů nalézá uplatnění v mnoha vědních disciplínách, zejména diskrétní matematice (teorie grafů, teorie uspořádaných množin a svazů) a algoritmizaci (grafové algoritmy, struktury dat). Rozsáhlé použití mají grafy také v aplikacích těchto věd. Například softwarové inženýrství (grafy volání procedur, hierarchie tříd v objektově orientovaném programování), databáze, informační systémy, systémy podporující rozhodování, umělá inteligence. Další aplikace lze nalézt i například v medicíně nebo biologii. Navíc bylo napsáno velké množství titulů věnujících se tématu automatického generování zobrazení grafů a diagramů a teoriím grafů a uspořádaných množin.

V této práci se zabývám speciálními grafy, používanými v diskrétní matematice, speciálně v algebře a ve vědních oborech ji využívajících jako základní matematický aparát. Jedná se o Hasseovy diagramy uspořádaných množin a svazů. Tyto základní algebraické struktury bylo, je a bude vždy potřeba jednoduše a rychle zobrazovat, a jejich nejlepší a nejuniverzálnější forma vizualizace je právě dobře vytvořený Hasseův diagram. V posledních letech pak kvůli rozvoji formální konceptuální analýzy výrazně roste potřeba grafické reprezentace svazů.

Diagramy uspořádaných množin a svazů s malým počtem prvků lze sice kreslit ručně, ale s narůstajícím počtem prvků je takové ruční kreslení diagramu velmi únavné a náročné. Je tedy žádoucí automatické generování počítačovými prostředky. Bohužel neexistuje univerzální metoda pro generování diagramu. Různým zaměřením a využití diagramů vyhovují různé metody. Existuje několik algoritmů, ale žádný neposkytuje obecně uspokojivé řešení. Není totiž zcela jasné, které vlastnosti tvoří dobrý diagram. Měl by být transparentní a snadno čitelný. Jak toho dosáhnout závisí v individuálních případech na zaměření interpretace diagramu a na vizuálních vlastnostech diagramu. Automatické generování mnohdy dává ne moc uspokojivé výsledky, nicméně jsou velmi užitečné v tom, že slouží jako velmi užitečný startovací bod pro vylepšení diagramu ručně.

Vytvoření dobrého diagramu svazu s více jak dvaceti prvky již vyžaduje praktické zkušenosti. Ale pro více jak 50 prvků i pečlivě zkonstruovaný diagram ztrácí na čitelnosti. Pro uspokojivé grafické reprezentace takových a větších svazů se využívají tzv. vnořené (nested) diagramy. Pro svazy čítající stovky prvků již není kompletní grafická reprezentace možná, v těchto případech je nutné aplikovat techniky rozdělení a reprezentování podsvazů.

Pro automatické generování diagramů svazů a vůbec softwarovou podporu zvláště konceptuálních svazů samozřejmě existují kvalitní nástroje a programy. Řekl bych, že v této oblasti dominuje software TOSCANA. *TOSCANA* (`http://www.mathematik.tu-darmstadt.de/ags/ag1/software/toscanademo/welcome_de.html`) je program, který umožňuje interakci s konceptuálními daty založenými na formálních kontextech skládajících se ze vztahů mezi objekty a atributy. Dalším zástupcem v této oblasti může být program *ANACONDA*. Programů by se našlo určitě více, jejich velkou nevýhodou je většinou uzavřenost a v nejhorších případech nejsou vůbec zveřejněny.

Já se v této práci orientuji spíše na zobrazování a generování diagramů obecných uspořádaných množin a svazů. V této oblasti už moc programů není, protože většina se věnuje právě nějaké konkrétní aplikaci těchto struktur, například zmíněné konceptuální svazy v konceptuální analýze dat. Hlavním zástupcem je zde program *LatDraw* (`http://www.math.hawaii.edu/~ralph/latdraw/`), jehož autorem je Ralph Freese. Tento program je určen pro generování diagramů svazů.

Cílem mé diplomové práce je implementace a srovnání metod automatického generování a zobrazování právě obecných uspořádaných množin a svazů, podobně jako je tomu u výše zmíněného programu. Diagram musí být generován pouze z definičních informací uspořádané množiny nebo svazu, tedy jejích prvků a vlastní relace uspořádání. Různých a různě kvalitních metod je téměř nepřeberné množství, takže v programu jsou implementovány jen nejvýraznější reprezentanti z tohoto množství. Vygenerovaný diagram by měl být co nejčitelnější a pokud možno by mohl odrážet i strukturu svazu. Samozřejmě žádná metoda nevytváří vždy dokonalé diagramy, takže program byl měl umožňovat i jeho manuální dotvoření a vylepšení podle představ uživatele.

Po tomto úvodu následuje nezbytná kapitola teoretických základů z oblasti uspořádaných množin a svazů a také jsou zde ujasněny pojmy z teorie grafů, které se používají při popisu metod generování diagramu.

V kapitole 3. jsou nejdříve rozebrány možné datové reprezentace uspořádaných množin a svazů. Význam správné reprezentace dat totiž nemůže zůstat opomenut. Samozřejmě jsou reprezentace důkladně porovnány. Potom již následuje stěžejní část popisu jednotlivých metod generování diagramu uspořádaných množin a svazů. Rozebrány jsou všechny metody implementované v programu, nemůže chybět jejich hodnocení a porovnání výsledných diagramů. Také jsou nastíněny další možné metody.

Další kapitola 4. již popisuje samotný program, požadavky na něj kladené, jeho strukturu, programátorský popis základních tříd a uživatelský popis grafického uživatelského rozhraní programu. Také se zmíním o možnostech exportu diagramu do různých grafických formátů.

V závěru je provedeno celkové shrnutí práce a naznačeny další možná rozšíření a vylepšení programu.

2. Teoretické základy

Analýza problému generování diagramu uspořádané množiny rozvedená v sekci 3. využívá mocný teoretický aparát uspořádaných množin a svazů a teorie grafů. Následující paragrafy proto slouží pro vymezení a sjednocení pojmů, používaných v dalším textu.

S pojmy intuitivní teorie množin se setkáváme již na střední škole a proto budu předpokládat znalost pojmů jako *množina a její prvek*, *množinová inkluze*, *systém množin*, *sjednocení*, *průnik a rozdíl množin*, *uspořádaná dvojice prvků*, *kartézský součin množin a kartézská mocnina množiny*. Dále předpokládám znalost pojmů ***relace na množině*** a ***množina s relací*** a ***zobrazení množiny do množiny*** a pojmů s nimi souvisejících jako *relace mezi množinami*, *složená relace a zobrazení*, *inverzní relace a zobrazení*, *reflexivní*, *symetrická*, *antisymetrická*, *tranzitivní a úplná relace*, *reflexivní a tranzitivní uzávěr relace*, *injektivní (prosté)*, *surjektivní (zobrazení na) a bijektivní (vzájemně jednoznačné) zobrazení*.

V následujících paragrafech o uspořádaných množinách a svazech jsou definovány základní pojmy, které jsou používány dále v textu, zejména v podsekci 3.2. o metodách generování diagramu uspořádaných množin. Také se zde hojně používá víceméně intuitivních pojmů z teorie grafů, které jsou upřesněny v části o grafech. Důležité věty jsou zde uvedeny důkazů, ty lze v případě potřeby nalézt v literatuře. Při psaní zbytku sekce jsem vycházel z [12], [5], [7], [1] a [3], kde lze rovněž nalézt užitečné věty a příklady, a z přednášek RNDr. Dagmar Skalské z předmětu Algebra.

2.1. Uspořádané množiny

V této podsekci zavedu pojmy **uspořádaná množina** a **Hasseův diagram uspořádané množiny**, pojmy s těmito související a nejdůležitější vlastnosti prvků těchto množin.

Definice 2.1. Nechť $\langle M, R\rangle$ je množina s relací, která je reflexivní a tranzitivní. Pak se relace R nazývá **kvaziuspořádání** a $\langle M, R\rangle$ se nazývá **kvaziuspořádaná množina**.

Je-li navíc relace R antisymetrická, nazývá se pak **(částečné) uspořádání** a $\langle M, R\rangle$ se nazývá **(částečně) uspořádaná množina**, neboli **poset (partially ordered set)**.

Je-li navíc relace R úplná, nazývá se pak **lineární uspořádání** a $\langle M, R\rangle$ se nazývá **lineárně uspořádaná množina** nebo krátce **řetězec**.

Označení 2.1. Relaci uspořádání budu v dalším textu značit symbolem $\leq$ (čti „menší nebo rovno").

Místo $x \leq y$ budu podle potřeby též psát $y \geq x$ (obojí znamená, že prvek x je v relaci uspořádání s prvkem y).

Pro $x \leq y \ \wedge \ x \neq y$ budu používat stručného označení $x < y$ (nazývaného *ostrá nerovnost*), nebo též $y > x$ (čti „x je menší než y“).

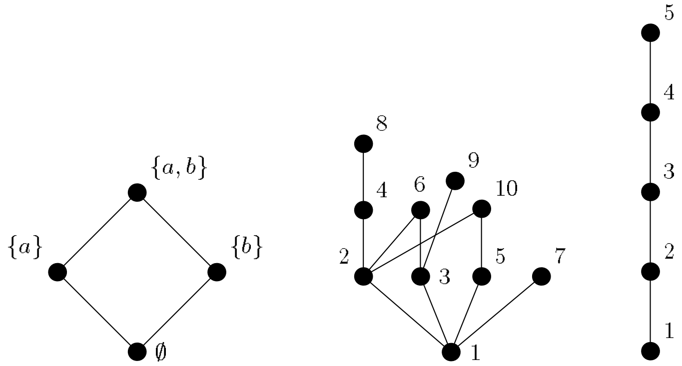
Definice 2.2. Dva prvky $x, y \in M$ se nazývají **srovnatelné**, jestliže $x \leq y$ nebo $y \leq x$. V opačném případě se prvky x, y nazývají **nesrovnatelné (paralelní)**, což značíme $x \parallel y$.

Definice 2.3. Nechť $M \neq \emptyset$ a $\leq$ je relace uspořádání na M, $x, y \in M$. Řekneme, že x je **předchůdce** y (x je **pokryto** y), nebo že y je **následovník** x (y **pokrývá** x), jestliže $x \leq y$ a pro každý prvek $w \in M$ takový, že $x \leq w \leq y$, je $x = w$ nebo $y = w$.

Relace $\prec \subseteq \leq$, tvořená pouze uspořádanými dvojicemi (předchůdce, následovník), se nazývá **relace pokrytí** relace $\leq$ (**relace pokrytí** na množině M).

Uspořádanou množinu $\langle M, \leq \rangle$ můžeme (pouze, je-li M konečná) graficky znázorňovat následujícím způsobem: prvky množiny M znázorníme jako body v rovině tak, aby v případě, že je $x < y$, ležel bod x níže než bod y. Dva body $x, y \in M$ spojíme úsečkou právě tehdy, když $x \prec y$.

Výsledný orientovaný graf (viz podsekce 2.3.) se pak nazývá **Hasseův (Hasseovský) diagram** uspořádané množiny $\langle M, \leq \rangle$. Množina vrcholů tohoto grafu je rovna množině M, množina hran je rovna množině relace pokrytí $\prec$. Dodávám, že uvedená konstrukce nedefinuje jednoznačně tvar Hasseova diagramu. Jednu a tutéž uspořádanou množinu je často možné znázornit Hasseovy diagramy různých tvarů tak, že na první pohled nemusí být vůbec zřejmé, že jde o diagramy téže uspořádané množiny. Na druhé straně, známe-li Hasseův diagram uspořádané množiny $\langle M, \leq \rangle$, pak z něj lze relaci $\leq$ jednoznačně zpětně zrekonstruovat. Konečnou uspořádanou množinu tedy můžeme zadávat Hasseovým diagramem. Nekonečné uspořádané množiny nelze Hasseovým diagramem reprezentovat, protože graf má podle definice konečný počet vrcholů. Můžeme jej pouze naznačit, třeba od nejmenšího prvku. Příklady Hasseových diagramů uspořádaných množin jsou na obrázku 1.



Obrázek 1. Hasseovy diagramy uspořádaných množin

Důsledek 2.1. *Každá konečná uspořádaná množina se dá vyjádřit Hasseovým diagramem.*

Definice 2.4. V uspořádané množině $\langle M, \leq \rangle$ se prvek $m \in M$ nazývá:

(i) **minimální**, jestliže neexistuje prvek $x \in M$ s vlastností $x < m$

(ii) **maximální**, jestliže neexistuje prvek $x \in M$ s vlastností $m < x$

(iii) **nejmenší**, jestliže pro $\forall x \in M$ platí $m \leq x$

(iv) **největší**, jestliže pro $\forall x \in M$ platí $x \leq m$

Věta 2.1. *Každá konečná uspořádaná množina $\langle M, \leq \rangle$ má alespoň jeden maximální a alespoň jeden minimální prvek.*

Poznámka. Předchozí věta obecně neplatí u nekonečných uspořádaných množin. Pokud tam platí, říkáme, že v této množině platí **minimální (resp. maximální) podmínka**. Např. otevřený interval $(0, 1)$ nemá ani maximum ani minimum, uspořádaná množina přirozených čísel $\langle \mathbb{N}, \leq \rangle$ má minimum, ale nemá maximum, ale zato uspořádaná množina podmnožin přirozených čísel $\langle 2^{\mathbb{N}}, \subseteq \rangle$ již má minimum ($\emptyset$) i maximum ($\mathbb{N}$).

Věta 2.2. *Nechť $\langle M, \leq \rangle$ je uspořádaná množina. Pak platí:*

1. *v $\langle M, \leq \rangle$ existuje nejvýše jeden nejmenší prvek a nejvýše jeden největší prvek*
2. *je-li $m \in M$ nejmenší (resp. největší) prvek, pak m je také minimální (resp. maximální) prvek a žádné další minimální (resp. maximální) prvky v uspořádané množině $\langle M, \leq \rangle$ neexistují*

Věta 2.3. *Nechť $\langle M, \leq \rangle$ je lineárně uspořádaná množina. Potom prvek $m \in M$ je minimální (resp. maximální) $\Leftrightarrow$ m je nejmenší (resp. největší).*

Definice 2.5. Nechť $\langle M, \leq \rangle$ je uspořádaná množina a $A \subseteq M$.

Dolním kuželem $\mathrm{L} \subseteq \mathrm{M}$ podmnožiny A v $\langle M, \leq \rangle$, značíme L(A), nazveme množinu všech prvků $m \in M$ takových, že pro $\forall x \in A$ platí $m \leq x$.

Horním kuželem $\mathrm{U} \subseteq \mathrm{M}$ podmnožiny A v $\langle M, \leq \rangle$, značíme U(A), nazveme množinu všech prvků $m \in M$ takových, že pro $\forall x \in A$ platí $x \leq m$.

Definice 2.6. Nechť A je neprázdná podmnožina uspořádané množiny $\langle M, \leq \rangle$.

Prvek $s \in A$ se nazývá:

a) **infimum** podmnožiny A v $\langle M, \leq \rangle$, značíme inf(A), jestliže je největším prvkem v $L(A)$

b) **supremum** podmnožiny A v $\langle M, \leq \rangle$, značíme sup(A), jestliže je nejmenším prvkem v $U(A)$

Definice 2.7. Nechť $\langle M, \leq \rangle$ je uspořádaná množina a $A \subseteq M$.

Podmnožinu A nazveme **podřetězec** v množině M, právě když $\langle A, \leq \rangle$ je lineárně uspořádaná množina.

Podmnožinu A nazveme **antiřetězec** v množině M, právě když každé dva prvky $x, y \in A$ jsou nesrovnatelné.

Řetězec (antiřetězec) A v množině M nazveme **maximální řetězec (antiřetězec)**, jestliže je maximální vzhledem k inkluzi.

Délka řetězce (antiřetězce) je počet jeho prvků.

Definice 2.8. **Výškou** uspořádané množiny $\langle M, \leq \rangle$, značenou h(M), rozumíme délku jejího maximálního řetězce.

Šířkou uspořádané množiny $\langle M, \leq \rangle$, značenou w(M), rozumíme délku jejího maximálního antiřetězce.

2.2. Svazy

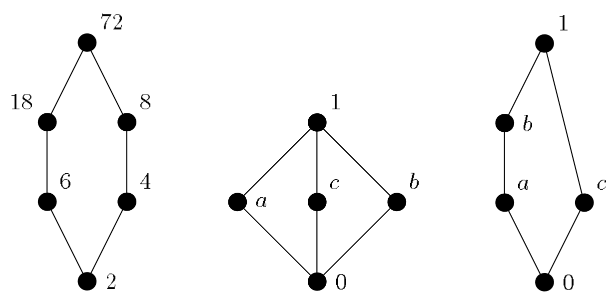
V této podsekci jsou zavedeny pojmy **svaz**, **Booleův svaz** a **Booleova algebra**, pojmy s těmito související a základní vlastnosti svazů.

Definice 2.9. Nechť $\langle A, \leq \rangle$ je uspořádaná množina.

Jestliže pro libovolnou dvojici prvků $a_1, a_2 \in A$ existuje $\sup\{\mathrm{a}_1, \mathrm{a}_2\}$, značené $a_1 \vee a_2$ (**spojení** a_1, a_2), a $\inf\{\mathrm{a}_1, \mathrm{a}_2\}$, značené $a_1 \wedge a_2$ (**průsek** a_1, a_2), pak se množina $\langle A, \leq \rangle$ nazývá **svaz** $\langle A, \vee, \wedge \rangle$.

Svaz nazveme **úplný**, existuje-li sup X a inf X pro libovolnou $X \subseteq A$.

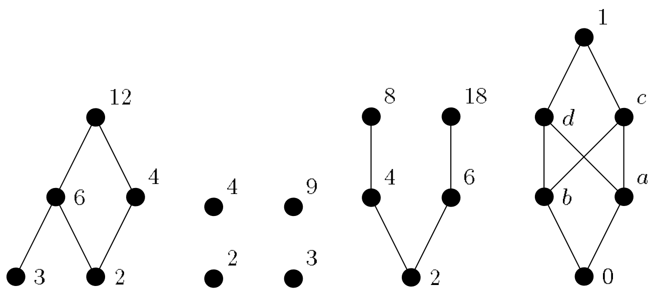
Na obrázku 2. lze nalézt příklady (Hasseových diagramů) svazů, na obrázku 3. potom příklady uspořádaných množin, jež nejsou svazy.



Obrázek 2. Svazy

Definice 2.10. Svaz $\langle A, \vee, \wedge \rangle$ nazveme **distributivní**, právě když pro $\forall a, b, c \in A$ platí:

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$$
$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$



Obrázek 3. Uspořádané množiny, jež nejsou svazy

Svaz $\langle A, \vee, \wedge \rangle$ nazveme **modulární**, právě když pro $\forall a, b, c \in A$ taková, že $a \leq c$, platí:

$$a \vee (b \wedge c) = (a \vee b) \wedge c$$

Věta 2.4. *Svaz $\langle A, \vee, \wedge \rangle$ je modulární, právě když pro $\forall a, b, c \in A$ platí:*

$$a \vee (b \wedge (a \vee c)) = (a \vee b) \wedge (a \vee c)$$

Věta 2.5. *Každý distributivní svaz je modulární.*

Definice 2.11. Nechť $\mathbb{A} = \langle A, \vee^A, \wedge^A \rangle$ je svaz a $B \subseteq A$.
Pak $\mathbb{B} = \langle B, \vee^B, \wedge^B \rangle$ se nazývá **podsvaz** svazu $\mathbb{A}$, jestliže $\forall a, b \in B$ platí:

$$a \wedge^B b = a \wedge^A b$$
$$a \vee^B b = a \vee^A b$$

Věta 2.6. *V každém konečném svazu existuje největší a nejmenší prvek.*

Poznámka. Uspořádaná množina, která má nejmenší a největší prvek, ale nemusí být nutně svaz, viz. množina úplně vpravo na obrázku 3.

Definice 2.12. Nechť $\langle A, \vee, \wedge \rangle$ je svaz.
Pokud existuje prvek $l \in L$ tak, že pro $\forall a \in L : a \vee l = a$, pak prvek l značíme 0 a nazýváme jej **nula svazu**. Pokud existuje prvek $u \in L$ tak, že pro $\forall a \in L : a \wedge u = a$, pak prvek u značíme 1 a nazýváme jej **jednička svazu**.
Svaz obsahující nulu a jedničku označujeme $\langle A, \vee, \wedge, 0, 1 \rangle$.

Definice 2.13. Nechť $\langle A, \vee, \wedge, 0, 1 \rangle$ je svaz s nulou a jedničkou, a $x \in A$.
Prvek $\overline{x} \in A$, pro který platí $x \vee \overline{x} = 1$ a $x \wedge \overline{x} = 0$, nazveme **doplněk (komplement)** prvku $x \in A$.
Svaz $\langle A, \vee, \wedge, 0, 1 \rangle$, v němž ke každému prvku $x \in A$ existuje aspoň jeden doplněk $\overline{x} \in A$, nazýváme **komplementární svaz**.

Definice 2.14. Distributivní komplementární svaz se nazývá **Booleův svaz**.

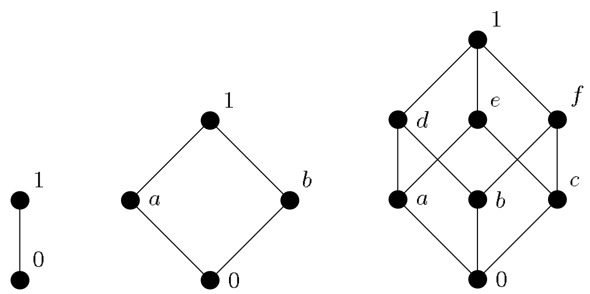
Věta 2.7. *V libovolném Booleově svazu pro všechny jeho prvky a, b platí:*

$$\overline{(a \vee b)} = \overline{a} \wedge \overline{b}$$
$$\overline{(a \wedge b)} = \overline{a} \vee \overline{b}$$

Definice 2.15. Nechť $\langle A, \vee, \wedge, 0, 1\rangle$ je Booleův svaz, pro jehož neutrální prvky platí $0 \neq 1$ a nechť $-$ označuje unární operaci komplementace. Šestice $\mathcal{A} = \langle A, \vee, \wedge, -, 0, 1\rangle$ se nazývá **Booleova algebra**.

Definice 2.16. Nechť $\mathcal{A} = \langle A, \vee, \wedge, -, 0, 1\rangle$ je Booleova algebra.
Prvek $a \in A, a \neq 0$ takový, že pro $\forall x \in A$ platí $x \wedge a = a$ nebo $x \wedge a = 0$, nazveme **atom** Booleovy algebry $\mathcal{A}$.

Poznámka. Hasseovy diagramy konečných Booleových algeber mají přirozenou geometrickou reprezentaci v podobě n-rozměrných krychlí, kde n je počet atomů algebry. Na obrázku 4. jsou Booleovy algebry s jedním, dvěma a třemi atomy.



Obrázek 4. Booleovy algebry s jedním, dvěma a třemi atomy

2.3. Grafy

V podsekci 3.2. o metodách generování diagramu uspořádaných množin se kromě pojmů z teorie grafů používají i některé základní pojmy z teorie vektorových prostorů, jako např. *vektor*, o nichž předpokládám, že je s nimi čtenář dostatečně obeznámen. V případě potřeby odkazuji na [5] nebo [9]. Zbytek podsekce je věnován **teorii grafů**, její obsah jsem čerpal z [3].

(Neorientovaný) graf G je uspořádaná dvojice $\langle V, E\rangle$, kde V je konečná množina *vrcholů* a $E \subseteq \{\{u, v\}; u, v \in V\}$ je konečná množina *(neorientovaných) hran*, sestávající z (neuspořádaných) dvojic $\{u, v\}$ vrcholů. Vrcholy grafu se někdy nazývají *uzly*.

Hrana $\{u, u\}$ se nazývá *smyčka*. *Jednoduchý graf* je graf, který nemá žádné smyčky.

Koncové vrcholy hrany $e = \{u, v\}$ jsou vrcholy u a v, říkáme, že vrcholy u a v jsou *sousední*. *Stupeň* vrcholu je počet jeho sousedních vrcholů (sousedů).

Orientovaný graf je definován stejně jako (neorientovaný) graf, s tím rozdílem, že prvky množiny $E \subseteq V \times V$, nazývané *orientované hrany*, jsou uspořádané dvojice $\langle u, v \rangle$ vrcholů. Orientovaná hrana $\langle u, v \rangle$ je *výstupní hrana* vrcholu u a *vstupní hrana* vrcholu v. *Vstupní* (resp. *výstupní*) *stupeň vrcholu* je počet jeho vstupních (resp. výstupních) hran.

(Orientovaná) cesta v (orientovaném) grafu $G = \langle V, E \rangle$ je posloupnost $\langle v_1, v_2, \ldots, v_n \rangle$ různých vrcholů G takových, že $\langle v_i, v_{i+1} \rangle \in E$ pro $1 \leq i \leq n-1$. (Orientovaná) cesta je *(orientovaný) cyklus*, jestliže $\langle v_n, v_1 \rangle \in E$. Orientovaný graf je *acyklický*, pokud nemá žádné orientované cykly.

Hrana $\langle u, v \rangle$ orientovaného grafu je *tranzitivní*, pokud existuje orientovaná cesta z vrcholu u do vrcholu v, která neobsahuje hranu $\langle u, v \rangle$. *Tranzitivní uzávěr* G' orientovaného grafu G obsahuje hranu $\langle u, v \rangle$ pro každou cestu z vrcholu u do vrcholu v. Ve většině případů vyjadřuje orientovaný graf stejnou informaci jako jeho tranzitivní uzávěr. Navíc mohou tranzitivní hrany značně znepřehlednit celý graf. Proto je obecně lepší kreslit *redukovaný orientovaný graf* (nebo také *tranzitivní redukci*), tj. orientovaný graf bez tranzitivních hran.

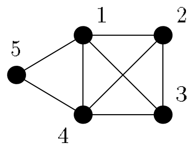
Graf $G' = \langle V', E' \rangle$ takový, že $V' \subseteq V$ a $E' \subseteq E \cap (V' \times V')$, je *podgraf* grafu $G = \langle V, E \rangle$. Pokud je $E' = E \cap (V' \times V')$, pak podgraf G' je *indukován* podmnožinou vrcholů V'.

Graf $G = \langle V, E \rangle$ s n vrcholy může být popsán *maticí sousednosti* A rozměru $n \times n$, jejíž řádky a sloupce odpovídají vrcholům a $A_{uv} = 1$, když $\langle u, v \rangle \in E$, a $A_{uv} = 0$ jinak. Tabulka 1. obsahuje takový popis grafu.

	1	2	3	4	5
1	0	1	1	1	1
2	1	0	1	1	0
3	1	1	0	1	0
4	1	1	1	0	1
5	1	0	0	1	0

Tabulka 1. Matice sousednosti grafu zobrazeného na obrázku 5.

Zobrazení Γ grafu (orientovaného grafu) G je prosté zobrazení každého vrcholu v grafu G do bodu $\Gamma(v)$ a každé hrany $\langle u, v \rangle$ do jednoduché otevřené křivky $\Gamma(u, v)$, s koncovými body $\Gamma(u)$ a $\Gamma(v)$. Orientovaná hrana je obvykle zobrazena jako šipka. Na obrázku 5. je zobrazení grafu, jehož matice sousednosti je v tabulce 1. Je důležité poznamenat, že graf a jeho zobrazení jsou dva odlišné objekty. Obecně může mít jeden graf více různých zobrazení a naopak dvěma různým grafům může odpovídat stejné zobrazení. Nicméně běžně se používá stejná terminologie pro hranu a zobrazení hrany. Např. tvrzení „hrana $\langle u, v \rangle$ je přímá čára“ je interpretováno jako „zobrazení $\Gamma(u, v)$ hrany $\langle u, v \rangle$ je přímá čára“.



Obrázek 5. Zobrazení grafu s maticí sousednosti v tabulce 1.

Zobrazení grafu je *planární*, jestliže se žádné dvě hrany nekříží. Graf je *planární*, jestliže umožňuje planární zobrazení. Planární zobrazení rozdělí rovinu do topologicky spojených regionů nazývaných *oblasti*. Neohraničené oblasti se nazývají *externí oblasti*.

Graf je *souvislý*, jestliže pro každé dva vrcholy u a v existuje v grafu cesta mezi těmito vrcholy. Maximální souvislý podgraf grafu G je *souvislá komponenta* grafu G.

Dělicí vrchol grafu G je vrchol, po jehož odstranění bude graf G nesouvislý. Souvislý graf bez dělicích vrcholů je *2-souvislý*. Maximální 2-souvislé podgrafy grafu jsou jeho *bloky* (někdy také nazývané *2-souvislé komponenty*). Platí, že graf je planární, právě když jeho bloky jsou planární. *Dělicí pár* $\langle u, v \rangle$ vrcholů 2-souvislého grafu je pár vrcholů, po jejichž odstranění bude graf nesouvislý. 2-souvislý graf bez dělicích párů vrcholů je *3-souvislý*. Uvedené pojmy souvislosti grafu si čtenář jistě dovede zobecnit na *k-souvislost grafu*.

Na závěr poznamenám, že pro každý orientovaný graf můžeme zkonstruovat *základní neorientovaný graf* tak, že neuvažujeme orientaci hran. To umožňuje převést terminologii (neorientovaných) grafů na orientované.

3. Rozbor problému

Příbuzenské struktury sestávající z množiny entit a vztahů mezi těmito entitami se nejčastěji modelují jako *grafy* (teorie viz podsekce 2.3.), entity jsou *vrcholy*, vztahy potom *hranami* tohoto grafu. Hasseovy diagramy uspořádaných množin jsou také grafy (viz jejich definice v podsekci 2.1. na straně 4). Diagramy se typicky kreslí s textem u vrcholu, vyjadřujícím jméno prvku množiny, a hrany diagramu jsou zobrazeny jako lomené čáry propojující vrcholy.

Ale dříve než můžeme přistoupit k samotnému zobrazování libovolného grafu, musíme mít nějakým způsobem uloženu jeho strukturu. U Hasseových diagramů vychází jeho struktura přímo z uspořádané množiny, proto musíme datově reprezentovat tuto uspořádanou množinu.

Následující podsekce proto pojednává o možných datových reprezentacích uspořádaných množin. Další podsekce pojednává o samotných metodách zobrazení Hasseova diagramu uspořádané množiny a svazu (dále jen diagramu uspořádané množiny nebo svazu).

3.1. Datová reprezentace

Efektivní datová reprezentace uspořádané množiny je velice důležitá, protože metody generování zobrazení diagramu hojně využívají při své práci nejrůznější operace a dotazy na uspořádaných množinách. Zdaleka nejčastěji používaným dotazem je zjištění vztahu dvou prvků, tedy zda je první menší než druhý. Proto tato operace musí být co nejrychlejší. Dále se často používají operace dotazující se na následovníky nebo předchůdce určitého prvku. Výjimkou také není zjištění minimálních či maximálních prvků z nějaké podmnožiny prvků uspořádané množiny. Tyto operace ale v cyklech používají základní operaci dotazu na vztah dvou prvků.

Zvolená datová reprezentace tedy přímo ovlivňuje rychlost a efektivitu práce s uspořádanou množinou a potažmo také samotné generování vzhledu diagramu. U vhodného uložení množiny ale nejde jen o rychlost práce s ní. Zvláště u velkých množin s mnoha prvky (řádu stovek) vyvstává další požadavek - prostorová efektivita. Pro dosažení co nejmenšího uložení množiny se využívá toho, že stačí uložit redukovanou podobu množiny, tzv. *relaci pokrytí*.

Paměťové nároky množiny ale úzce souvisejí s časovými. Při uložení relace pokrytí nejsou uloženy reflexivní a tranzitivní vztahy prvků. Při základním dotazu na vztah dvou prvků se ale ve většině případů stává, že zjišťované prvky nejsou přímo srovnatelné, jejich vazba tedy vede přes několik jiných prvků. Dotaz tedy vede na iteraci, při které se vyhledávají tyto „propojovací“ prvky tranzitivního vztahu.

Na druhou stranu uložení všech reflexivních a tranzitivních vztahů s cílem zvýšení rychlosti dotazů na vztahy prvků množiny zvyšuje paměťové nároky této množiny.

3.1.1. Dvourozměrné pole

Datová reprezentace pomocí dvourozměrného pole je realizována jako tabulka relace uspořádání množiny, v níž řádky i sloupce představují prvky množiny a v políčku tabulky na řádku prvku x a sloupci prvku y je nenulové číslo, pokud $x \leq y$, jinak políčko obsahuje číslo 0.

Z předchozího je vidět, že v tabulce jsou uloženy nejenom vztahy relace pokrytí, ale i reflexivní a tranzitivní vztahy. Tyto nadbytečné informace můžeme ukládat, protože velikost reprezentace to nijak neovlivní. Tabulka má konstantní velikost $n \times n$ políček, kde n je počet prvků uspořádané množiny, a všechny vztahy mezi prvky se ukládají do této tabulky. Je tedy naopak výhodné tyto vztahy ukládat, protože se při dotazech na vztah prvků nemusejí nijak složitě zjišťovat (platí samozřejmě hlavně pro tranzitivní vztahy).

Operace uspořádané množiny na takto reprezentovaných datech jsou velice rychlé, protože můžeme přímo přistupovat ke konkrétnímu políčku pole a zjistit tak vztah (i tranzitivní) dvou prvků. Základní operace porovnání dvou prvků má v ideálním případě konstantní časovou složitost $O(1)$, v praxi se ovšem projeví nepatrná závislost na počtu prvků (zvláště u většího množství prvků) způsobená konkrétním uložením tabulky v paměti.

Opačná situace je ovšem u paměťové náročnosti této reprezentace. Tabulka má konstantní velikost $n \times n$ políček, nezávisle na počtu vztahů mezi prvky. Paměťová složitost této reprezentace dat je tedy $O(n^2)$. U řídkých uspořádání bude většina políček nevyužitých a nadbytečných, nicméně přítomných a zabírajících místo. Hustá uspořádání naopak dané paměťové místo plně využijí. Kromě samotné tabulky vztahů mezi prvky množiny je ale potřeba uložit také samotné prvky, kterých je n. Nároky na paměť tedy vzrostou o $O(n)$.

3.1.2. Seznam seznamů

Datová reprezentace seznam seznamů je podobná předchozí reprezentaci pomocí tabulky relace uspořádání. Rozdíl je v tom, že se pro každý prvek množiny ukládají pouze jeho následovníci nebo předchůdci. Reprezentace tedy představuje seznam seznamů následovníků nebo předchůdců.

Tedy zde se neukládají nadbytečné reflexivní a tranzitivní vztahy, které by pouze prodlužovaly seznamy prvků evidované pro každý prvek množiny. Ukládá se pouze relace pokrytí, tedy jenom nezbytné informace o množině, nic navíc. To ale znamená, že při drtivé většině dotazů na vztahy mezi prvky v množině se musí tranzitivní vztahy znovu a znovu ze seznamů zjišťovat několikanásobným procházením těchto seznamů.

Operace na takto reprezentované množině jsou pomalé, protože se musí nejdříve projít seznam prvků a potom seznam jeho následovníků nebo předchůdců. Tento průchod je v seznamu sekvenční a tedy nejpomalejší možný. Počet prvků množiny je n, počet prvků v seznamu jeho následovníků nebo předchůdců může být také v nejhorším případě až n. Časová složitost základní operace zjištění vztahu mezi dvěma prvky množiny je tedy $O(n^2)$, a to ještě jenom u dvojic prvků (předchůdce, následovník). U tranzitivních vztahů je časová náročnost ještě vyšší kvůli zjišťování prvků na cestě z menšího prvku do většího prvku.

Paměťová náročnost této reprezentace je naopak velice příznivá, protože ukládáme jenom nezbytně nutné informace. První je samozřejmě samotný seznam prvků, kterých je n. Další jsou již informace o vztazích těchto prvků, tj. relace pokrytí. Tyto informace představují seznam následovníků nebo předchůdců pro každý prvek množiny, který v praxi obsahuje malý konstantní počet prvků. Dá se tedy říct, že paměťová složitost této reprezentace je lineární, $O(n)$.

3.1.3. Binární vyhledávací strom

U této datové reprezentace ukládáme uspořádané dvojice prvků (předchůdce, následovník) z relace uspořádání množiny. Jedná se tedy v podstatě o stejné uložení jako v případě seznamu seznamů, pouze s tím rozdílem, že každý prvek množiny je uložen tolikrát, kolik má předchůdců nebo následovníků. Dvojice se ukládají do binárního vyhledávací stromu, seřazeného podle prvního nebo druhého prvku dvojice.

Stejně jako u seznamu seznamů se neukládají nadbytečné reflexivní a tranzitivní vztahy. Tedy se ukládá opět jen relace pokrytí. Tranzitivní vztahy se musí opětovně zjišťovat.

Časová náročnost této reprezentace je ale oproti předchozí reprezentaci pomocí seznamu seznamů výrazně lepší. Vyhledávání v binárním vyhledávacím stromu se děje v logaritmickém čase, a ve stromě máme uloženy přímo dvojice prvků, u nichž chceme zjistit jejich vztah. Časová složitost zjištění vztahu dvojice prvků (předchůdce, následovník) je tedy $O(\log\ n)$. U tranzitivních vztahů je vyšší kvůli zjišťování prvků na cestě z menšího prvku do většího prvku, podobně jako u seznamu seznamů.

Paměťová náročnost už je ale téměř identická s paměťovou náročností reprezentace pomocí seznamu seznamů. Ukládá se pouze relace pokrytí ve své původní podobě, tj. uspořádané dvojice prvků. Její velikost je v praxi úměrná počtu prvků v uspořádané množině, paměťová složitost této reprezentace je tedy, stejně jako v případě seznamu seznamů, $O(n)$. Oproti seznamu seznamů ukládáme sice samotné prvky množiny několikrát (ve všech dvojicích s tímto prvkem), ale tento násobek je v praxi konstantní (viz odstavec v části 3.1.2. o reprezentaci pomocí seznamu seznamů).

3.1.4. Srovnání

Při srovnání výše uvedených reprezentací dat jsou na první pohled zřejmé některé rozdíly v časových i paměťových složitostech těchto reprezentací. Reprezentace pomocí dvourozměrného pole je velice rychlá, zato má ale velké paměťové nároky, zvláště pak u velkých množin. Reprezentace pomocí seznamu seznamů a pomocí binárního vyhledávacího stromu jsou naopak celkem pomalé, ale zato vyžadují jen nezbytně nutnou paměť. Časová náročnost vyhledávání ve stromu je sice logaritmická, ale v praxi se to oproti seznamu seznamů projeví až u větších množin.

Z pohledu časové náročnosti se jako jednoznačně nejvýhodnější jeví reprezentace pomocí dvourozměrného pole, a to jak pro malé, tak i pro středně velké i pro velké uspořádané množiny. Vyplývá to z jejího v podstatě konstantního času zjištění vztahu dvou prvků. Druhé dvě reprezentace jsou oproti této prostě příliš pomalé. Pokud nám tedy záleží na rychlosti práce s množinou, musíme volit tuto reprezentaci dat.

U paměťových nároků jednotlivých reprezentací je to už složitější. Pro malé množiny je opět výhodná reprezentace pomocí dvourozměrného pole, kde paměťová náročnost ještě není tak velká. Pro větší a velké množiny už ale tyto nároky na paměť nezadržitelně rostou, se čtvercem počtu prvků. Naopak druhé dvě reprezentace můžeme směle použít i pro velké množiny, protože jejich nároky na paměť rostou jen lineárně s počtem prvků.

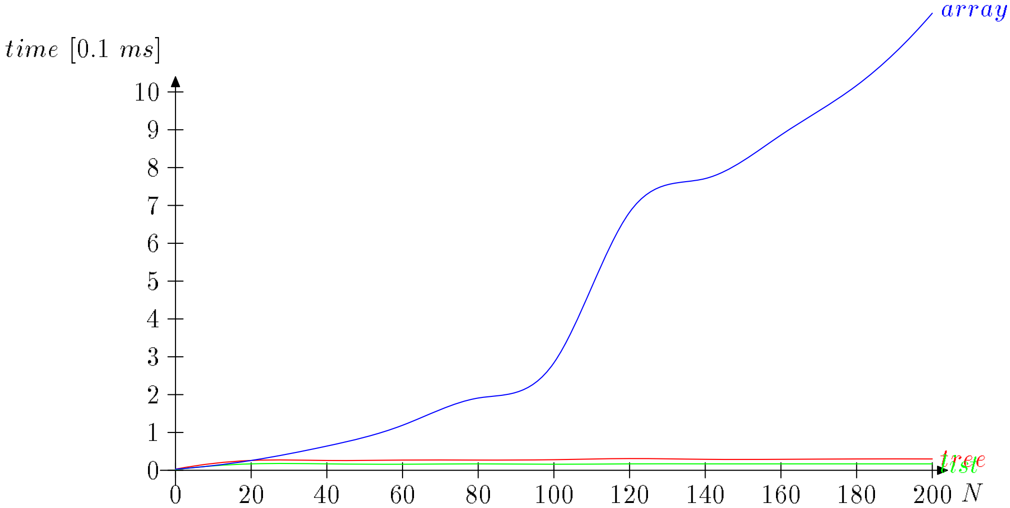
Pro ukládání středně velkých a velkých uspořádaných množin by tedy bylo dobré vytvořit nějakou kombinaci přístupů pole a seznamu nebo stromu. V ukázkových příkladech této práce se pracuje spíše s menšími množinami, čítajícími tak do 50-ti prvků, proto jako nejlepší reprezentace vychází dvourozměrné pole.

Na následujících obrázcích jsou různé výkonnostní i paměťové grafy srovnávající všechny datové reprezentace při provedení stěžejních operací na uspořádané množině. Na vodorovné ose je vždy počet prvků množiny při provedení operace, na svislé potom čas spotřebovaný provedením operace nebo paměť spotřebovaná po provedení operace. Modrou čarou je značen průběh výsledků operace pro dvourozměrné pole, zeleně pro seznam seznamů a červeně pro reprezentaci pomocí vyhledávacího stromu. Pro případ černobílého zobrazení je průběh označen i slovně (postupně *array*, *list* a *tree*).

Na obrázku 6. je graf průměrné doby vložení nového prvku do množiny prvků čítající N prvků. Čas vložení nového prvku při reprezentaci pomocí seznamu seznamů nebo pomocí stromu je nepatrný a konstantní v porovnání s reprezentací pomocí pole. To není principiální rozdíl, to je dáno implementací změny velikosti dvourozměrného pole. Alokace paměti pro novou tabulku a kopírování obsahu původní tabulky do nové trvá hodně času.

Obrázek 7. zobrazuje průměrný čas zaznamenání vztahu dvou prvků do množiny čítající N prvků. Čas pro dvourozměrné pole není konstantní, protože záleží na konkrétním uložení pole v paměti (spojité či fragmentované) a také na faktu, že se musí doplňovat tranzitivní vztahy. Čas vložení nové dvojice prvků do vyhledávacího stromu je zřetelně menší než čas vyhledání prvku v seznamu a vložení prvku do seznamu, protože do stromu vkládáme přímo dvojici prvků.

Na obrázku 8. vidíme graf zobrazující průměrný čas porovnání dvou prvků, mezi nimiž jsou přímé i tranzitivní vztahy, v množině s N prvky. Čas pro dvou-



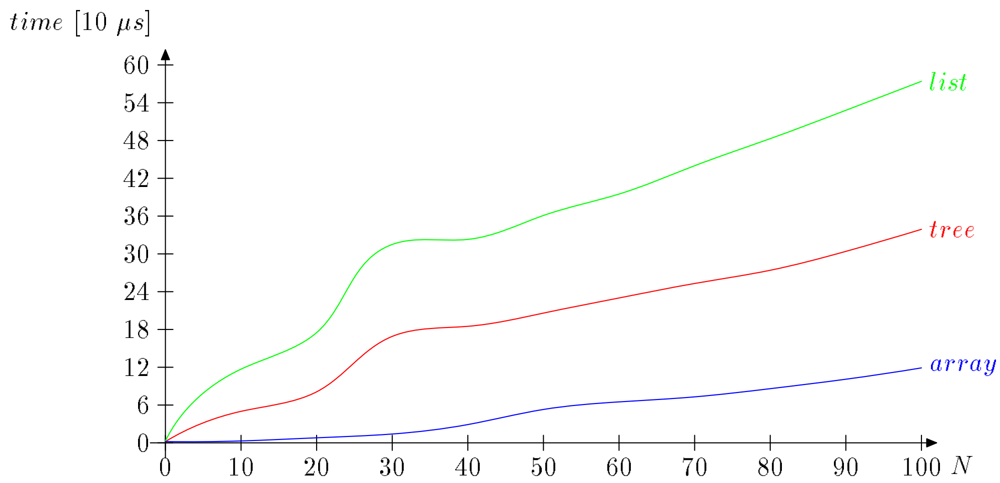
Obrázek 6. Průměrný čas vložení nového prvku do množiny s N prvky

rozměrné pole je téměř konstantní, důvody, proč není zcela konstantní viz výše. Zároveň je naprosto nepatrný v porovnání s ostatními dvěma reprezentacemi. Čas pro reprezentace pomocí seznamu seznamů a stromu je téměř stejný z důvodu výkonné implementace seznamu pomocí binárního vyhledávacího stromu, viz konkrétní popis implementace v části 4.3.1.

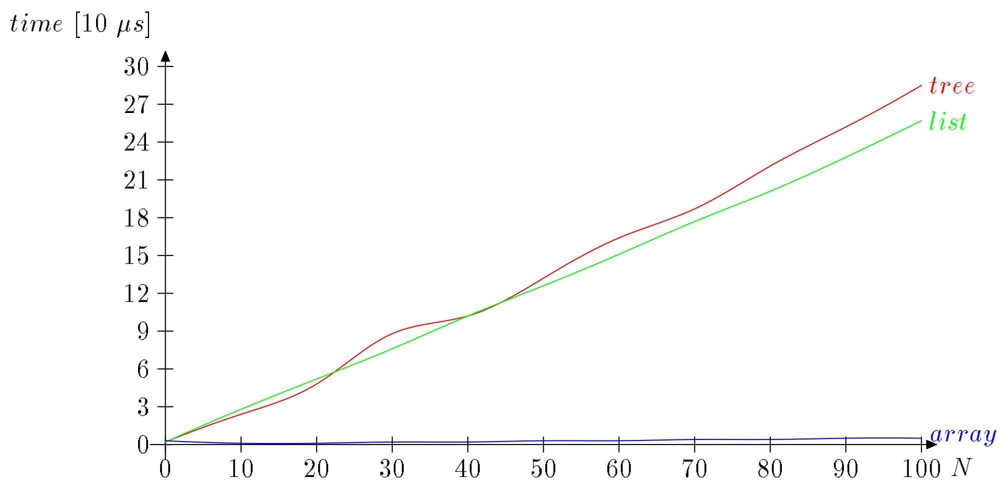
Na obrázku 9. je graf celkové doby provedení operací zjištění následovníků každého prvku, vyhledání maximálních prvků z celé množiny, určení největšího prvku z celé množiny a vyhledání dolního kužele z podmnožiny obsahující pouze největší prvek, za předpokladu N prvků v množině. Čili graf prezentující praktické použití množiny. Je jasně vidět, že ostatní dvě reprezentace jsou v porovnání s dvourozměrným polem v praxi příliš pomalé, již od 20 prvků výše. Navíc jejich časové nároky rostou příliš rychle.

Další obrázek 10. zobrazuje graf spotřebované paměti po uložení N prvků do množiny. Je vidět, že paměť u reprezentace pomocí seznamu seznamů a stromu roste lineárně s počtem vložených prvků a je v podstatě stejná. Naopak paměťová náročnost u reprezentace pomocí pole roste kvadraticky s počtem prvků a pro více jak 1000 prvků (mimo oblast grafu) již roste nade všechny meze! Tato reprezentace je paměťově výhodnější vždy do určité hranice určené implementací (zde asi 130 prvků).

Na posledním obrázku 11. vidíme spotřebu paměti navíc od vložení N prvků po uspořádání prvků každý s každým (pokud je to možné). U dvourozměrného pole by to mělo být přísně konstantní (a nulové), ale projevuje se zde režie paměti kódu programu a zmenšující se rozdíl při větším celkovém obsazení paměti programem. Seznam seznamů a strom mají stejný průběh navýšení spotřeby paměti, strom spotřebuje nepatrně více, protože ukládá každý prvek několikrát (viz rozbor výše), ale při větším počtu prvků se již stává výhodnějším, viz konec grafu.



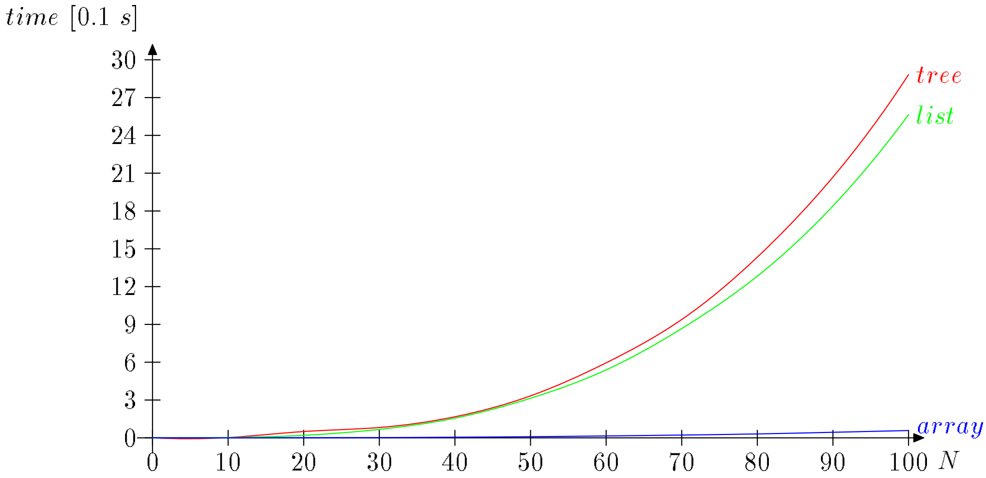
Obrázek 7. Průměrný čas zaznamenání vztahu dvou prvků do množiny s N prvky



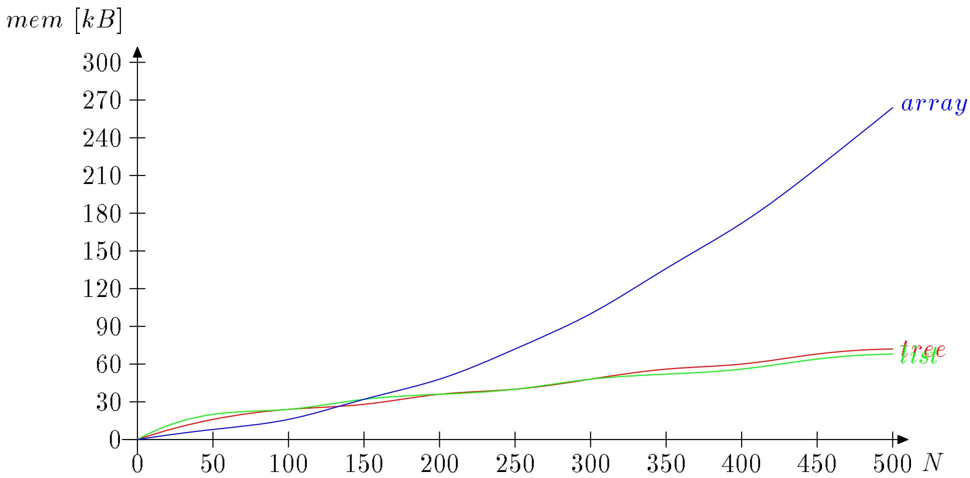
Obrázek 8. Průměrný čas porovnání dvou prvků v množině s N prvky

3.2. Metody generování diagramu

Tato podsekce obsahuje rozbor problematiky generování zobrazení diagramu uspořádaných množin. V části 3.2.1. jsou rozebrány parametry různých metod, konvence kreslení diagramu a kritéria na jeho vzhled a další požadavky, které musí dobré zobrazení diagramu splňovat. Tato část víceméně vychází z obecnější problematiky kreslení grafů, obsáhle prezentované v [3]. V části 3.2.2. je uveden „intuitivní“ postup kreslení diagramu, v částech 3.2.3. až 3.2.5. jsou detailně prezentovány v programu implementované metody generování zobrazení diagramu. V další části jsou výsledky používaných metod podrobeny hodnocení a porovnání. Závěrečná část podsekce potom uvádí krátký závěr popisu těchto metod a stručný popis dalších možných metod generování zobrazení diagramu uspořádaných množin.



Obrázek 9. Celkový čas provedení několika operací na množině s N prvky

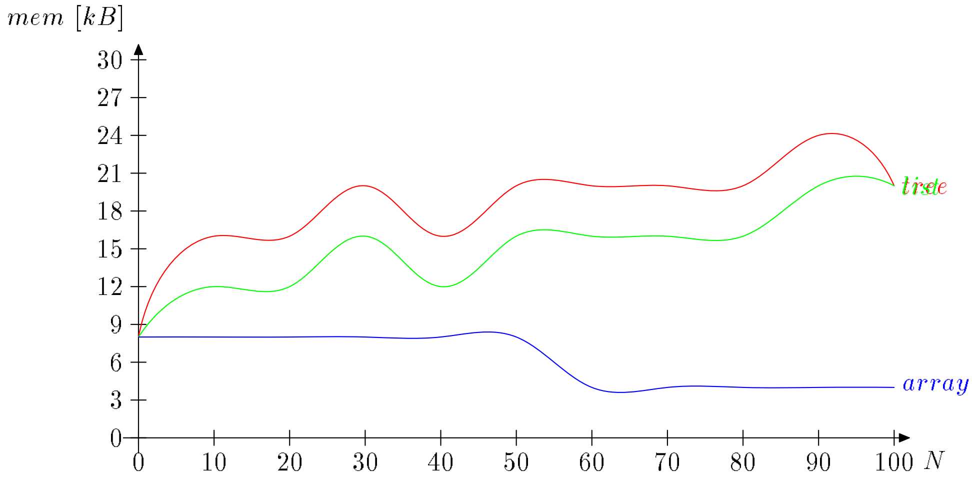


Obrázek 10. Paměť potřebná pro uložení N prvků do množiny

V následujícím textu se *generováním diagramu* rozumí generování zobrazení orientovaného grafu ve smyslu uvedeném v podsekci 2.3. o grafech. Orientovaným grafem se zde myslí právě Hasseův diagram uspořádané množiny. Jedná se tedy o dosud v textu používané generování zobrazení Hasseova diagramu. Pro zjednodušení budu v následujícím textu generováním Hasseova diagramu myslet generování tohoto zobrazení.

Ještě dodám, že orientované hrany Hasseových diagramů se nekreslí jako šipky, ale jen jako lomené čáry, jako kdyby hrany byly neorientované. Orientace hran mezi vrcholy je u tohoto diagramu dána uspořádáním prvků množiny reprezentovaných těmito vrcholy a v samotném diagramu navíc také polohou těchto vrcholů (viz definice Hasseova diagramu v podsekci 2.1. na straně 4).

Na závěr tohoto úvodu je důležité zdůraznit, že se diagram ve všech dále



Obrázek 11. Paměť spotřebovaná navíc po uspořádání N prvků v množině

popsaných metodách generuje pouze z uspořádané množiny, tzn. vstupem všech metod je pouze výčet prvků množiny a vlastní relace uspořádání. Metody neuvažují a ani nezachovávají již existující, ručně vytvořený nebo předchozí generovaný, diagram!

3.2.1. Parametry metod a požadavky na ně

Potřeba různých parametrů metod generování diagramu vyvstává ze zjištění, že „nejlepší" diagram nemusí existovat (a v drtivé většině případů ani neexistuje, vždy totiž záleží na interpretaci diagramu podle oblasti, kterou modeluje). Různé uspořádané množiny a svazy mají různé strukturální vlastnosti a proto mohou vyžadovat různý přístup k vytvoření dobrého diagramu. Proto se zavádí důležité koncepty kreslení diagramu uvedené dále.

Konvence zobrazení jsou základní pravidla, která musí diagram splňovat, aby byl akceptovatelný. Například konvence pro Hasseův diagram říkají, že vrcholy se zobrazují jako puntíky nebo malé kroužky, u nichž je jméno prvku množiny, hrany jako lomené nebo přímé čáry. Běžně používané konvence jsou zobrazování hran jako lomených (někdy i jako jednoduchých otevřených křivek) nebo přímých čar, ortogonální kreslení (čáry hran jsou buď horizontální nebo vertikální), kreslení do mřížky, planární zobrazení, kreslení v určitém směru (horizontálním i vertikálním) a další.

Požadavky na vzhled specifikují vlastnosti diagramu, které bychom rádi obsáhli v co největší míře pro dosažení co největší čitelnosti diagramu. Běžně adoptované požadavky jsou minimalizace celkového počtu křížení hran (ideálně planární diagram), minimalizace plochy diagramu, minimalizace délky hrany, minimalizace celkového počtu lomení čar a počtu lomení jedné čáry, zobrazení symetričností v diagramu a další. Vyjmenované požadavky vedou přirozeně k optimalizačním problémům, které jsou mnohdy výpočetně velmi složité. Proto pro ně bylo navrženo mnoho aproximačních strategií a heuristik.

Některé výše zmíněné požadavky na vzhled ale působí navzájem proti sobě, určité kompromisy jsou tedy nevyhnutelné. A i když přijaté požadavky nejsou v konfliktu, je často algoritmicky složité si poradit se všemi zároveň.

Další požadavky se mohou aplikovat na dílčí pod-diagramy diagramu nebo na umístění jednotlivých vrcholů diagramu. Jsou to například požadavky na umístění vrcholu co nejvíce do středu diagramu nebo naopak co nejvíce na kraj diagramu, umístění nějaké podmnožiny vrcholů blízko sebe (tzv. klastrování vrcholů), umístění vrcholů na cestě v horizontálním nebo vertikálním směru, nebo i třeba předdefinování určitého tvaru diagramu.

Nakonec samozřejmě záleží i na výkonnosti algoritmu kreslení. Interaktivní aplikace vyžadují okamžitou odezvu, i u větších množin. Výkonnost je tedy důležitá pro praktické použití metod generování diagramu.

Z výše uvedeného vyplývá, že různé metody generování diagramu si stanoví různé priority požadavků. Tyto priority některým uspořádaným množinám a svazům vyhovují, jiným zase ne. Přístupy prezentované v následujících částech často rozdělují celý proces vytvoření diagramu na sekvenci určitých kroků, každý s cílem uspokojit nějakou podmnožinu požadavků.

3.2.2. Nejjednodušší přímá metoda

Ještě před popisem implementovaných metod načrtnu postup na první pohled zřejmě nejjednodušší metody přímého kreslení diagramu. Tento postup by asi bez delšího přemýšlení napadl každého čtenáře těchto řádků.

- Nejdříve nakreslíme vrchol největšího prvku. Pod něj umístíme vrcholy jeho předchůdců.

- Teď systematicky tvoříme infima prvků již nakreslených vrcholů, nebo místo tvoření infim můžeme znovu umístit vrcholy předchůdců prvků s již umístěnými vrcholy, pod ně vrcholy jejich předchůdců, atd. Tvoříme tak vlastně diagram „po vrstvách“.

- Nakonec doplníme vrcholy zbývajících prvků.

Uvedený postup lze samozřejmě provádět i z „opačného konce“, tj. od prvku nejmenšího. Místo předchůdců potom samozřejmě uvažujeme následovníky a místo infim suprema.

Tato metoda se může na první pohled zdát velice jednoduchá, musíme si však uvědomit, že zde stále zůstává problém *konkrétního rozmístění vrcholů prvků diagramu.* To můžeme sice provést „intuitivně“, ale pak bude nejspíše potřeba více iterací postupu pro vytvoření uspokojivého diagramu, který stále ještě nemusí dosahovat kvalit diagramů produkovaných dále uvedenými metodami. Proto se tento postup používá nejvíce u ručního kreslení diagramů velmi malých uspořádaných množin a svazů, kde tak máme proces vytvoření diagramu plně pod kontrolou.

„Skutečné“ metody generování diagramu řeší právě i toto konkrétní rozmístění vrcholů prvků a proto přejdeme k jejich popisu.

3.2.3. Úrovňová metoda

Tato „metoda“ je můj vlastní postup na kreslení diagramu uspořádané množiny. Vymyslel jsem ji před jakýmkoliv studiem literatury věnující se tomuto tématu a vychází víceméně z výše nastíněné nejjednodušší přímé metody generování diagramu. Metoda tedy nemá žádné hlubší teoretické základy, snaží se pouze vhodně uspořádat vrcholy ve vrstvách (zde nazývaných „úrovně“) tak, aby bylo dosaženo co možná nejmenšího celkového počtu křížení hran a diagram vypadal vizuálně co možná nejlépe (v rámci možností tohoto velice jednoduchého postupu). Metodu může čtenář a uživatel programu brát jako jakýsi „amatérský“ pokus o řešení jinak velice rozsáhlé problematiky generování diagramu uspořádané množiny.

Postup by se dal v bodech vyjádřit takto:

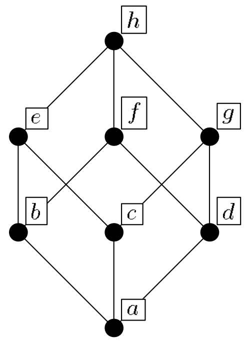
- Nejprve umístíme na stejnou úroveň vrcholy všech minimálních prvků uspořádané množiny v libovolném pořadí, např. v pořadí, v jakém jsou prvky vyjmenovány přímo v uspořádané množině. Všechny vrcholy zleva ohodnotíme číslem jejich pořadí na úrovni (číslováno od 0).

- Do další (vyšší) úrovně umístíme vrcholy všech následovníků prvků z předchozí úrovně. Pokud již pro nějakého následovníka existuje v diagramu vrchol, nahradíme jej novým vrcholem v nynější nové úrovni. Vrcholy v úrovni rozmístíme rovnoměrně a souměrně podle středu předchozí úrovně. Vrcholy prvků umisťujeme v pořadí, v jakém získáme následovníky prvků předchozí úrovně, které bereme postupně podle jejich pořadí. Pořadí více následovníků jednoho prvku na předchozí úrovni je libovolné. Teď přistoupíme k ohodnocení nově umístěných vrcholů. Vrchol prvku na nové úrovni ohodnotíme součtem ohodnocení vrcholů všech jeho předchůdců. Následně vrcholy uspořádáme (vzestupně) zleva doprava podle jejich právě vzniklého ohodnocení a ještě upravíme ohodnocení všech těchto vrcholů (opět zleva doprava) tak, aby hodnoty tvořily rostoucí posloupnost. Všechny sousední vrcholy spojíme čarou.

• Předchozí bod opakujeme, dokud nejsou umístěny vrcholy všech prvků.

Předchozí postup lze samozřejmě pozměnit tak, aby se diagram generoval „shora“, tzn. začneme od umístění vrcholů maximálních prvků a postupujeme po úrovních dolů, pracujíc s předchůdci. Z uvedeného postupu je vidět, že generuje diagram obecné uspořádané množiny, není tedy zaměřen třeba jen na svazy.

Jako demonstrační příklad uvedu postup generování diagramu jednoduchého diagramu Booleovy algebry se třemi atomy (jejíž diagram tvoří z geometrického pohledu třírozměrnou krychli). Výsledný diagram je zobrazen na obrázku 12.



Obrázek 12. Diagram osmi-prvkového Booleova svazu vygenerovaný úrovňovou metodou

Nejprve umístíme vrchol nejmenšího prvku a (jako jediného minimálního). Ohodnotíme jej číslem 0.

Na další úroveň rozmístíme vrcholy prvků b, c a d, rovnoměrně a souměrně podle vrcholu nejmenšího prvku pod nimi (čili vrchol prostředního prvku c nad něj a vrcholy ostatních dvou prvků na kraj). Ohodnocení vrcholů je součet ohodnocení všech jejich předchůdců. V tomto případě ale máme u všech tří prvků pouze jednoho předchůdce a jeho vrchol má ohodnocení rovno 0, tedy vrcholy všech tří prvků a, b i c ohodnotíme také číslem 0. Uspořádávat je nemusíme (všechny mají stejné ohodnocení), pouze upravíme jejich ohodnocení na rostoucí posloupnost, tedy vrcholu prvku a zůstane číslo 0, vrchol prvku prvek b bude mít 1 a vrchol prvku c číslo 2. Vrcholy prvků spojíme čarou s vrcholem nejmenšího prvku.

Na další úrovni už to bude zajímavější. Následovníci prvků z předchozí úrovně jsou prvky e, f a g. Jejich vrcholy umístíme přímo nad vrcholy jejich předchůdců. Vrchol prvku e bude mít ohodnocení $0 + 1 = 1$, prvku f potom $0 + 2 = 2$ a vrchol prvku g obdrží ohodnocení $1+2 = 3$. Vrcholy prvků jsou již také uspořádány podle ohodnocení, které ani nemusíme měnit, protože tvoří rostoucí posloupnost. Vrcholy opět spojíme čarou s vrcholy jejich předchůdců.

Poslední úroveň obsahuje vrchol posledního, největšího prvku množiny, h. Na úrovni jej umístíme doprostřed diagramu. Nyní již z čistě formálních důvodů jej ohodnotíme číslem $1 + 2 + 3 = 6$ a nakonec spojíme čarou s vrcholy jeho předchůdců.

3.2.4. Vrstvová metoda

Vrstvová metoda reprezentuje hierarchický přístup pro kreslení acyklických orientovaných grafů, ve kterém jsou vrcholy rozvrženy do vrstev ve vertikálním nebo horizontálním směru a hrany jsou kresleny pomocí lomených čar. Problematika je dostatečně podrobně rozebrána v [3], z této publikace také vychází většina rozboru této metody uvedeného dále. Další možná rozšíření a vylepšení následujících postupů, např. vedení čar hran tak, aby neprotínaly vrcholy, lze nalézt v [6]. Diagram uspořádané množiny je acyklický orientovaný graf, můžeme na něj proto hierarchický přístup přímo aplikovat. Hierarchický přístup obsahuje následující tři kroky:

- *Rozvržení do vrstev* rozmisťuje vrcholy do horizontálních vrstev a určuje tak jejich y-souřadnici.
- *Redukce křížení* uspořádává vrcholy v každé vrstvě tak, aby počet křížení hran byl co nejmenší.
- *Přiřazení horizontálních souřadnic* přiřazuje konečné x-souřadnice vrcholům při zachování pořadí těchto vrcholů, vypočítané v předchozím kroku. Na konci tohoto kroku obdržíme konečné zobrazení grafu. V tomto kroku mohou být uplatněny některé požadavky na vzhled zobrazení grafu, např. snížení zakřivení (zalomení) hran nebo horizontální rozmístění vrcholů pro dosažení symetričnosti.

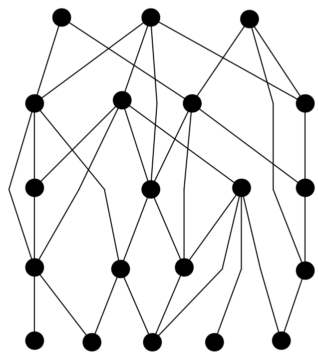
Rozvržení do vrstev

Nejdříve je potřeba zavést nutnou terminologii. Předpokládejme, že $G = \langle V, E \rangle$ je acyklický orientovaný graf. *Rozvržení do vrstev* je rozklad na množině vrcholů V do podmnožin $L_1, L_2, \dots, L_h$ tak, že pokud $\langle u, v \rangle$ je hrana, kde vrchol $u \in L_i$ a vrchol $v \in L_j$, pak $i < j$. Acyklický orientovaný graf s rozvržením do vrstev je *vrstvový orientovaný graf*. *Výška* vrstvového grafu je počet h jeho vrstev. *Šířka* vrstvového grafu je počet vrcholů ve vrstvě s maximálním počtem vrcholů, tj. $\max_{1 \leq i \leq h} |L_i|$. Délka hrany $\langle u, v \rangle$, kde vrchol $u \in L_i$ a vrchol $v \in L_j$, je $j - i$. Vrstvový graf je *korektní*, pokud žádná hrana není delší jak jedna.

Vrstvové grafy adoptují konvenci kreslení po vrstvách, kde každý vrchol ve vrstvě L_i má y-souřadnici rovnu i, viz obrázek 13.

Na rozvržení do vrstev se kladou tři požadavky:

1. Rozvržení do vrstev by mělo být kompaktní, to znamená, že výška a šířka by měly být malé. Vzdálenost mezi vrstvami je konstantní.
2. Rozvržení do vrstev by mělo být korektní. Toho dosáhneme tak, že hrany $\langle u, v \rangle$ delší jak jedna ($k > 1$) nahradíme cestami $\langle u = v_1, v_2, \dots, v_k = v \rangle$, kde vrcholy $v_2, \dots, v_{k-1}$ jsou tzv. *falešné vrcholy*. Tyto falešné vrcholy jsou potřeba, protože krok redukce křížení hran předpokládá, že graf je korektní.



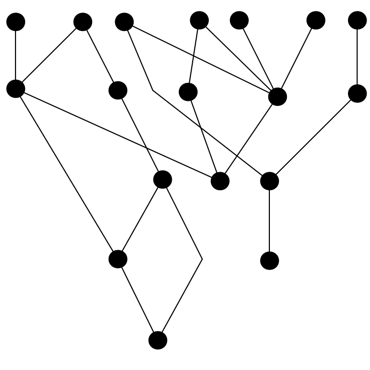
Obrázek 13. Konvence kreslení diagramu po vrstvách

3. Počet falešných vrcholů by měl být malý. Je proto několik důvodů: čas spotřebovaný jednotlivými kroky rozvržení do vrstev závisí na celkovém počtu vrcholů, skutečných i falešných, čáry představující zobrazení hran jsou lomeny jen ve falešných vrcholech a čitelnost diagramu klesá s narůstajícím počtem lomení čar, a v neposlední řadě je pro člověka jednodušší sledovat kratší cesty (hrany) než delší.

V následujících odstavcích jsou představeny dvě metody rozvržení do vrstev, první minimalizuje výšku, ale ignoruje šířku, druhá bere v úvahu výšku i šířku výsledného zobrazení. Existují samozřejmě i další metody rozvržení do vrstev, např. metoda minimalizující počet falešných vrcholů, realizovaná pomocí lineárního programování. Pro jejich popis již ale odkazuji na [3].

Nejdelší cesta

Nejdříve umístíme vrcholy všech maximálních prvků uspořádané množiny do vrstvy L_h. Pak umístíme každý zbývající vrchol do vrstvy L_{h-p}, kde p je délka nejdelší cesty z vrcholu do nějakého maximálního vrcholu. Příklad rozvržení do vrstev touto metodou je na obrázku 14.



Obrázek 14. Rozvržení vrcholů do vrstev metodou nejdelší cesty

Tato metoda rozvržení do vrstev má dvě atraktivní vlastnosti: může být provedena v lineárním čase, protože graf je acyklický, a vytváří minimální počet vrstev. Naopak největší nevýhodou je to, že výsledné zobrazení může být příliš široké. Například nejvyšší vrstva rozvržení na obrázku 14. je relativně široká.

Minimalizace šířky

Rozvržení do vrstev pomocí metody nejdelší cesty minimalizuje výšku. Nicméně kompaktnost výsledného zobrazení závisí i na šířce.

Naneštěstí problém nalezení rozvržení s minimální šířkou při zachování minimální výšky je NP-úplný (důkaz viz [3], provádí se převodem na NP-úplný problém plánování procesorů). Propojení s plánováním procesorů nabízí heuristiky pro rozvržení do vrstev. Popíši zde nyní *Coffman-Grahamův algoritmus rozvržení do vrstev*. Tento algoritmus pracuje kromě orientovaného grafu také s kladným číslem W představujícím maximální šířku rozvržení. Cílem tohoto algoritmu je také udržet výšku rozvržení co nejmenší.

Ještě je dobré poznamenat, že do šířky rozvržení se nepočítají falešné vrcholy, protože velikost těchto vrcholů je zanedbatelná v porovnání s velikostí skutečných vrcholů. Většinou se falešné vrcholy vůbec nezobrazují, představuje je pouze zlom čáry, propojující skutečné vrcholy.

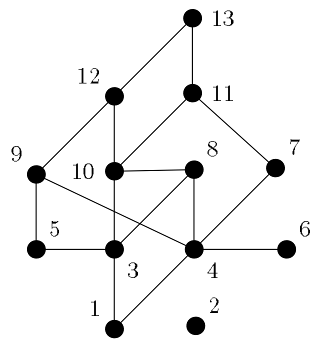
Algoritmus má dvě fáze: první uspořádá vrcholy a druhá je rozvrhne do vrstev. Algoritmus používá uspořádání definované na konečných množinách kladných čísel následujícím způsobem. Pokud je S konečná množina kladných čísel, potom $\max(S)$ je největší prvek S. Pak $S < T$ pokud platí jedna z následujících podmínek:

1. $S = \emptyset$ a $T \neq \emptyset$, nebo
2. $S \neq \emptyset$, $T \neq \emptyset$, a $\max(S) < \max(T)$, nebo
3. $S \neq \emptyset$, $T \neq \emptyset$, $\max(S) = \max(T)$, a $S - \{\max(S)\} < T - \{\max(T)\}$.

Jedná se zde vlastně o jednoduché lexikografické uspořádání, kde je nejvýznamnější největší prvek množiny, např. $\{1, 4, 7\} < \{3, 8\}$ nebo $\{3, 4, 9\} < \{1, 5, 9\}$.

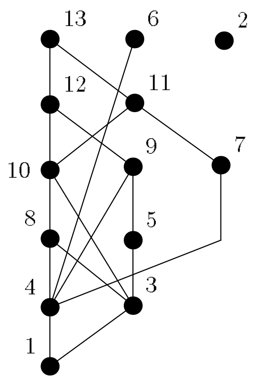
První fáze algoritmu uspořádá vrcholy přiřazením kladných čísel každému vrcholu. První číslo 1 je přiřazeno vrcholu nějakého minimálního prvku uspořádané množiny (libovolného). Další číslo v pořadí je přiřazeno vrcholu takového prvku, kterému ještě nebylo přiřazeno žádné číslo, vrcholům všech předchůdců prvku již bylo přiřazeno číslo a množina čísel přiřazených vrcholům předchůdců prvku je minimální ve smyslu lexikografického uspořádání $<$ definovaného výše. Příklad výstupu této procedury je na obrázku 15.

Druhá fáze Coffman-Grahamova algoritmu rozmístí vrcholy do vrstev s tím, že žádná vrstva nebude obsahovat více jak W vrcholů. Postupujeme od nejvyšší



Obrázek 15. Přiřazení čísel vrcholům první fází Coffman-Grahamova algoritmu

vrstvy L_h k nejnižší vrstvě L_1. Do vrstvy L_k umístíme vrchol, který ještě nebyl umístěn do nějaké vrstvy a vrcholy všech následovníků jeho prvku již byli umístěny do nějaké předchozí vyšší vrstvy $L_h, \ldots, L_{k+1}$. Pokud je zde více takových vrcholů, pak vybereme ten s největším přiřazeným číslem. Pokud naopak podmínky nesplňuje žádný vrchol anebo je již vrstva plná (tj. $|L_k| = W$), přesuneme se na další (nižší) vrstvu L_{k-1}. Rozmístění vrcholů s $W = 3$, vytvořené pro přiřazení čísel na obrázku 15., je ilustrováno na obrázku 16.



Obrázek 16. Rozvržení vrcholů do vrstev druhou fází Coffman-Grahamova algoritmu z očíslovaných vrcholů na obrázku 15.

Vložení falešných vrcholů

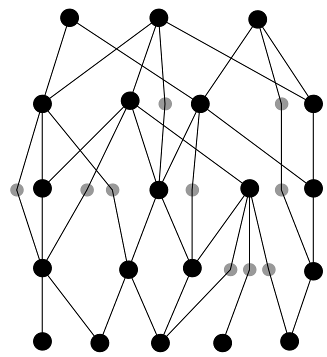
Po rozvržení do vrstev následuje vložení falešných vrcholů, protože následující krok, redukce křížení hran předpokládá na vstupu korektní graf.

Před samotným vkládáním falešných vrcholů ale ještě rozmístíme skutečné vrcholy ve vrstvách na konkrétní (předběžné) x-souřadnice. To proto, že falešné vrcholy vkládáme do vrstev také na konkrétní x-souřadnice, viz dále.

Jak již bylo řečeno, hrany $\langle u, v\rangle$, kde $u \in L_i$ a $v \in L_j$, $i < j$, delší jak jedna nahradíme cestami $\langle u = v_1, v_2, \ldots, v_n = v\rangle$, kde vrcholy $v_2, \ldots, v_{n-1}$ jsou falešné vrcholy. Tyto falešné vrcholy umisťujeme do vrstev $L_{i+1}, \ldots, L_{j-1}$. Aby byly cesty tvořené falešnými vrcholy co nejpřímější (chceme minimalizovat zakřivení lomené čáry hrany), umístíme každý falešný vrchol v_k ve vrstvě L_k na konkrétní x-souřadnici $x(v_k)$ podle vztahu:

$$x(v_k) = \frac{k-i}{j-i}(x(v) - x(u)) + x(u),$$

kde $x(u)$, resp. $x(v)$, je x-souřadnice vrcholu u, resp. vrcholu v. Vložení falešných vrcholů do vrstvového grafu ilustruje obrázek 17.



Obrázek 17. Vložení falešných vrcholů do vrstvového diagramu na obrázku 13.

Nyní již můžeme přistoupit k redukci křížení, protože po doplnění falešných vrcholů do grafu je tento korektní.

Redukce křížení

Počet křížení hran ve vrstvovém grafu nezávisí na přesných horizontálních pozicích vrcholů, pouze na pořadí těchto vrcholů v každé vrstvě. Tedy problém redukce křížení hran je kombinatorického charakteru a spočívá v nalezení vhodného pořadí vrcholů, ne geometrického, kde by šlo o určení správné x-souřadnice pro každý vrchol. I když nám to problém konceptuálně zjednodušuje, stále zůstává složitý. Po pravdě, problém minimalizace křížení hran ve vrstvovém grafu je NP-úplný, i když máme jenom dvě vrstvy. Zůstává NP-úplný, i když je v každé vrstvě pouze jeden skutečný vrchol. Proto byly vymyšleny různorodé heuristiky pro snížení počtu křížení. Právě tyto heuristiky jsou obsahem následujících odstavců.

Průchod vrstvami

Redukce počtu křížení hran probíhá tak, že zvolíme pevné pořadí vrcholů v první vrstvě L_1. Potom pro pevné pořadí vrcholů ve vrstvě L_{i-1}, $i = 2, 3, \ldots, h$, uspořádáváme vrcholy ve vrstvě L_i, abychom snížili počet křížení hran mezi vrstvami L_{i-1} a L_i. Existují i varianty tohoto základního postupu. Například můžeme zvolit pevné pořadí pro dolní i horní vrstvu L_{i-1} a L_{i+1} a přeuspořádávat vrcholy vrstvy L_i pro snížení počtu křížení hran zároveň mezi vrstvami L_{i-1}, L_i a L_i, L_{i+1}. Dále můžeme postupovat opačným směrem, od poslední vrstvy k první nebo postup několikrát v různých směrech opakovat, dokud bude počet křížení hran klesat.

Tento postup (a všechny jeho zmíněné varianty) ale předpokládá řešení následujícího problému. Za pevného pořadí vrcholů ve vrstvě L_{i-1} máme určit pořadí vrcholů ve vrstvě L_i tak, aby bylo dosaženo minimálního počtu křížení hran. Tento problém se nazývá *problém křížení mezi dvěma vrstvami*. Tento problém je fundamentální pro řešení celého problému křížení hran vrstvového grafu, jeho řešení je v literatuře věnována velká pozornost, toto řešení bude proto obsahem následujících odstavců.

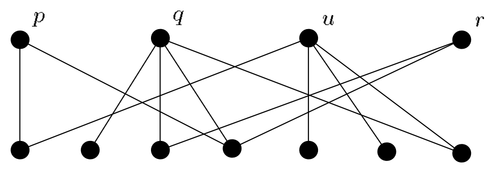
Dvouvrstvový orientovaný graf je orientovaný graf $G = \langle L_1, L_2, E\rangle$, kde L_1 a L_2 jsou disjunktní množiny vrcholů a $E \subseteq L_1 \times L_2$ je množina hran.

I když naším cílem je uspořádat vrcholy ve vrstvách, uvažujeme u každého vrcholu ve vrstvě L_i jeho x-souřadnici x_i zavedenou při vkládání falešných vrcholů. Počet křížení v zobrazení grafu G se souřadnicemi x_1 a x_2 vrcholů ve vrstvách je označen $\mathrm{cross}(G, x_1, x_2)$, a minimální počet křížení za předpokladu pevného pořadí vrcholů ve vrstvě L_1 daného souřadnicemi x_1 je označen $\mathrm{opt}(G, x_1)$. Platí:

$$\mathrm{opt}(G, x_1) = \min_{x_2} \mathrm{cross}(G, x1, x2).$$

Naneštěstí problém křížení mezi dvěma vrstvami je NP-úplný. Následují tedy dvě základní heuristické metody pro jeho řešení.

Pro obě metody můžeme vypozorovat následující důležité zjištění: pokud jsou u a v vrcholy ve vrstvě L_2, pak počet křížení mezi hranami incidentními s u a hranami incidentními s v závisí pouze na relativní pozici u a v a ne na pozici jiných vrcholů. Toto zjištění nás motivuje k zavedení čísla křížení. *Číslo křížení* c_{uv} je počet křížení, které tvoří hrany incidentní s u s hranami incidentními s v, kde $x_2(u) < x_2(v)$. Formálněji, pro $u \neq v \in L_2$, je c_{uv} počet párů $\langle u, w\rangle$, $\langle v, z\rangle$ hran, kde $x_1(z) < x_1(w)$. Pro úplnost definujeme $c_{uu} = 0$ pro všechny $u \in L_2$. Obrázek 18. zobrazuje dvouvrstvový orientovaný graf a tabulka 2. jemu odpovídající čísla křížení pro každý pár vrcholů v horní vrstvě.



Obrázek 18. Dvouvrstvový graf odpovídající tabulce 2. čísel křížení hran

Dodejme, že pomocí čísel křížení můžeme vypočítat $\mathrm{cross}(G, x_1, x_2)$ a dolní odhad pro $\mathrm{opt}(G, x_1)$, viz následující lemma.

	p	q	u	r
p	0	2	1	1
q	5	0	6	3
u	6	9	0	6
r	2	3	2	0

Tabulka 2. Tabulka čísel křížení hran pro vrcholy horní vrstvy grafu na obrázku 18.

Lemma 3.1. *Jestliže $G = \langle L_1, L_2, E \rangle$ je dvouvrstvový orientovaný graf a x_1 a x_2 jsou souřadnice udávající pořadí vrcholů ve vrstvách L_1 a L_2, pak*

$$\mathrm{cross}(G, x_1, x_2) = \sum_{x_2(u) < x_2(v)} c_{uv}. \tag{1}$$

Navíc platí

$$\mathrm{opt}(G, x_1) \geq \sum_{u,v} \min(c_{uv}, c_{vu}), \tag{2}$$

kde suma je přes všechny neuspořádané páry $\{u, v\}$ vrcholů horní vrstvy x_1.

Důkaz. Identita 1 je zřejmá. Nerovnost 2 vyplývá z toho, že pro každé pořadí vrcholů ve vrstvě L_2 (včetně optimálního pořadí) je buď $x_2(u) < x_2(v)$ nebo $x_2(v) < x_2(u)$. □

Třídící metody

Cílem řešení problému křížení mezi dvěma vrstvami je seřadit vrcholy ve vrstvě L_2 tak, aby se minimalizoval počet křížení. Tohle setřídění se dá udělat jednoduchými třídícími metodami, kdy třídíme vrcholy podle čísel křížení.

První je *třídění prohozením sousedů*, ve kterém prohazujeme sousední vrcholy s použitím čísel křížení, způsobem podobným známému bublinkovému třídění (Bubble sort). Procházíme vrstvu L_2 zleva doprava a prohodíme sousední vrcholy u a v, kdykoliv platí $c_{uv} > c_{vu}$. Tento průchod opakujeme, dokud nejsou prohozeny žádné vrcholy, tj. počet křížení neklesne. Jelikož c_{uv} závisí pouze na relativních pozicích vrcholů u a v, není potřeba měnit hodnoty čísel křížení. Můžeme tedy každý průchod vrstvou L_2 provést v čase $O(|L_2|)$ a počet průchodů je maximálně také $O(|L_2|)$. Časová složitost tohoto algoritmu je proto $O(|L_2|^2)$.

Můžeme také provést třídění metodou podobnou třídění rozdělováním (Quick sort), nazývanou *třídění rozdělením*. Vybereme vrchol $p \in L_2$ nazývaný *pivot* a každý vrchol $u \neq p \in L_2$ zatřídíme nalevo od vrcholu p, pokud je $c_{up} \leq c_{pu}$, jinak jej zatřídíme napravo od vrcholu p. Předchozí je rekurzivně aplikováno na množinu vrcholů nalevo a na množinu vrcholů napravo od vrcholu p. Výstupem algoritmu je spojení výstupu z aplikace na množinu vrcholů nalevo od p, pivotu

p a výstupu z aplikace na množinu napravo od vrcholu p. Algoritmus se samozřejmě provádí, pokud je množina vrcholů neprázdná, a toto je zároveň koncová podmínka rekurze. Nejhorší časová složitost je opět $O(|L_2|^2)$, ale v praxi běží (stejně jako třídění rozdělováním) v čase $O(|L_2| \log |L_2|)$.

Obě předchozí metody potřebují napřed výpočet čísel křížení a proto mají nelineární časovou složitost. Další dvě metody již mají lineární složitost.

Metoda průměru a metoda mediánu

V *metodě průměru* se, jednoduše řečeno, x-souřadnice každého vrcholu $u \in L_2$ nastaví na průměr x-souřadnic jeho sousedů. Tedy

$$x_2(u) = \mathrm{avg}(u) = \frac{1}{\deg(u)} \sum_{v \in N_u} x_1(v),$$

kde $\deg(u)$ značí stupeň vrcholu u a N_u je množina sousedů vrcholu u. Pokud je množina sousedů prázdná (a tedy stupeň vrcholu u je nulový), je $x_2(u) = 0$. Pokud mají dva vrcholy stejné hodnoty průměru, uspořádáme je libovolně.

Metoda mediánu je podobná metodě průměru. Jednoduše řečeno, x-souřadnice každého vrcholu $u \in L_2$ se nastaví na medián x-souřadnic jeho sousedů. Tady potřebujeme definovat pojem *medián*. Pokud sousední vrcholy vrcholu u jsou $v_1, v_2, \ldots, v_j$, kde $x_1(v_1) < x_1(v_2) < \cdots < x_1(v_j)$, pak definujeme medián $\mathrm{med}(u) = x_1(v_{j/2})$. Pokud vrchol u nemá sousední vrcholy, je $\mathrm{med}(u) = 0$. Pokud mají dva vrcholy stejné hodnoty mediánu a jeden má lichý stupeň a druhý sudý, potom vrchol s lichým stupněm zařadíme nalevo od vrcholu se sudým stupněm. Pokud je i parita stupňů vrcholů stejná, potom je seřadíme libovolně.

Není těžké dokázat, že obě metody průměru i mediánu dávají nulové křížení, pokud je nulové křížení možné. Nicméně ale žádná z těchto metod nedává optimální řešení.

V [3] lze nalézt i další metodu, realizovanou pomocí lineárního programování, jehož prostředky lze problém křížení dvou vrstev také řešit.

Další podrobnosti o všech předchozích metodách redukce křížení hran lze nalézt v [3].

Přiřazení horizontálních souřadnic

Poslední krok vrstvové metody kreslení diagramu přiřazuje konečné x-souřadnice vrcholům ve vrstvách při zachování pořadí těchto vrcholů, vypočítané v kroku redukce křížení. Problém můžeme nahlížet jako optimalizační problém, ve kterém se snažíme např. opět o napřímení lomených čar v diagramu podle stejného vzorce jako při vkládání falešných vrcholů, v nichž ke zlomu dochází. Při řešení tohoto optimalizačního problému může ale dojít ke zvětšení šířky diagramu. Pokud nám na šířce diagram záleží, musíme zavést další požadavky na takto řešené přiřazování souřadnic. Na konci tohoto kroku obdržíme konečné zobrazení grafu.

3.2.5. Geometrická metoda

Geometrická metoda je založena na představě teoretické struktury svazu skrz geometrickou reprezentaci a následně nalezení nejlepšího možného uspořádání diagramu. To znamená, že si nejdříve nakreslíme, od ruky nebo pomocí počítače, jakýsi pomocný obrázek, který použijeme k vytvoření skutečného diagramu. Tento pomocný obrázek se nazývá *geometrický diagram*. Když si svaz představíme jako tří-dimenzionální diagram a podíváme se na něj z jeho nejvyššího bodu, tj. největšího prvku, uvidíme právě geometrický diagram.

Shora nejprve uvidíme předchůdce největšího prvku. V geometrickém diagramu jsou reprezentovány kroužky, ve kterých jsou jména prvků. Pak pokračujeme ve vytváření geometrického diagramu podle těchto pravidel:

1. Prvek s právě jedním následovníkem je reprezentován kroužkem, který je částečně překrytý kroužkem tohoto následovníka.

2. Prvek s právě dvěma následovníky je reprezentován čarou mezi těmito většími prvky. Jméno prvku je zapsáno do kroužku, který je částečně překrytý touto čarou.

3. Prvek s právě třemi následovníky je reprezentován trojúhelníkem propojujícím tyto větší prvky. Jméno prvku je zapsáno do trojúhelníku.

Prvky s více jak třemi následovníky jsou reprezentovány analogicky n-úhelníkem propojujícím tyto následovníky. Největší a nejmenší prvky svazu nejsou v diagramu zakresleny.

Tímto způsobem můžeme dostat geometrický diagram jako na obrázku 19. Nyní zde uvedu konkrétní kroky k získání tohoto diagramu pro množinu, jejíž výčet prvků spolu se vztahy mezi nimi (ve formě tzv. *seznamu následovníků*, kde ke každému prvku jsou uvedeni jeho následovníci) je v tabulce 3.

Prvek 2 je předchůdce prvku 1, tedy kroužek pro prvek 2.

Prvek 3 je předchůdce prvku 2, tedy kroužek pro prvek 3, částečně překrytý kroužkem prvku 2.

Prvek 4 je předchůdce prvku 1, tedy kroužek pro prvek 4.

Prvek 5 je předchůdce prvků 2 a 4, tedy čára pro prvek 5 propojující kroužky prvků 2 a 4.

Prvek 6 je předchůdce prvků 3 a 5, tedy čára pro prvek 6 propojující kroužek prvku 3 a čáru prvku 5.

Prvek 7 je předchůdce prvku 1, tedy kroužek pro prvek 7.

Prvek 8 je předchůdce prvku 7, tedy kroužek pro prvek 8, částečně překrytý kroužkem prvku 7.

Prvek 9 je předchůdce prvků 2 a 7, tedy čára pro prvek 9 propojující kroužky prvků 2 a 7.

1:				
2:	1			
3:	2			
4:	1			
5:	2	4		
6:	3	5		
7:	1			
8:	7			
9:	2	7		
10:	9			
11:	3	9		
12:	4	7		
13:	8	12		
14:	5	9	12	
15:	10	14		
16:	6	11	14	
17:	6			
18:	13	15	16	17

Tabulka 3. Seznam následovníků prvků uspořádané množiny

Prvek 10 je předchůdce prvku 9, tedy kroužek pro prvek 10, částečně překrytý čarou prvku 9.

Prvek 11 je předchůdce prvků 3 a 9, tedy čára pro prvek 11 propojující kroužek prvku 3 a čáru prvku 9.

Prvek 12 je předchůdce prvků 4 a 7, tedy čára pro prvek 12 propojující kroužky prvků 4 a 7.

Prvek 13 je předchůdce prvků 8 a 12, tedy čára pro prvek 13 propojující kroužek prvku 8 a čáru prvku 12.

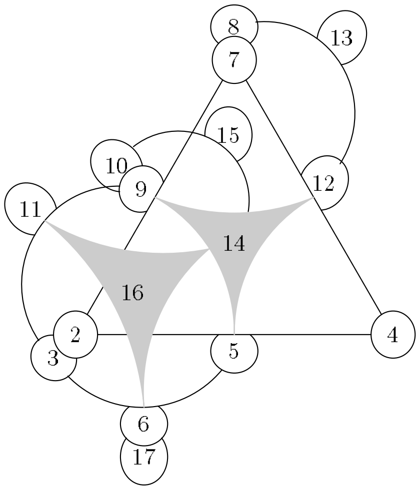
Prvek 14 je předchůdce prvků 5, 9 a 12, tedy trojúhelník pro prvek 14 propojující čáry prvků 5, 9 a 12.

Prvek 15 je předchůdce prvků 10 a 14, tedy čára pro prvek 15 propojující kroužek prvku 10 a trojúhelníku prvku 14.

Prvek 16 je předchůdce prvků 6, 11 a 14, tedy trojúhelník pro prvek 16 propojující čáry prvků 6, 11 a trojúhelníku prvku 14.

Prvek 17 je předchůdce prvku 6, tedy kroužek pro prvek 17, částečně překrytý čarou prvku 6.

Zbývá říct, jak z tohoto geometrického diagramu vytvořit dobrý diagram svazu. Diagram svazu, kterému odpovídá geometrický diagram na obrázku 19., je na obrázku 20. Z něj je vidět, že nejmenší podstruktura svazu sestává z dvou Booleovských svazů. Diagramu (krychle) tohoto podsvazu můžeme dosáhnout systematicky. Začneme od předchůdců největšího prvku, reprezentovaných nepřekrytými kroužky v geometrickém diagramu, v našem příkladě jsou to prvky 2,

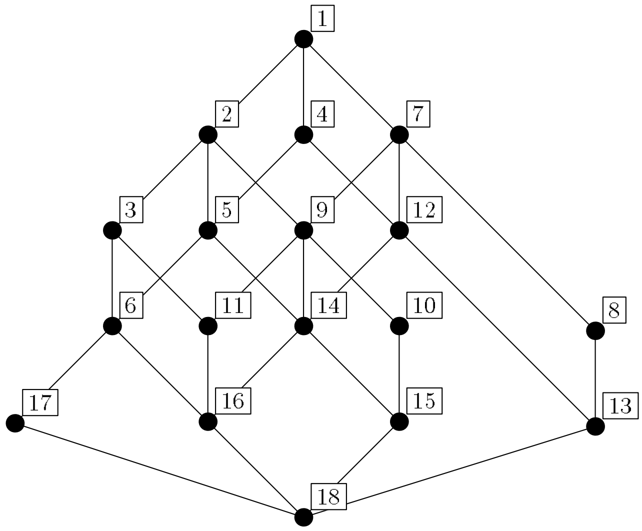


Obrázek 19. Geometrický diagram vytvářený geometrickou metodou

4 a 7. Kroužky těchto prvků jsou po dvou propojeny čarami prvků 5, 9 a 12, které jsou následně spojeny trojúhelníkem prvku 14. To znamená, že prvky 1, 2, 4, 7, 5, 9, 12 a 14 tvoří Booleovský podsvaz. Otázka je, jak těchto osm prvků v diagramu nejlépe rozmístit. Po nakreslení vrcholu největšího prvku se zdá být rozumné umístit vrcholy atomů 2 a 7 na kraj a vrchol prvku 4 mezi ně, protože pod vrcholy obou prvků 2 a 7 je další „bod“, pro který je potřeba nějaké místo. Vrcholy prvků 7, 5, 9 a 12 budou nejlépe umístěny podle *pravidla paralelogramů*, které říká, že vrchol prvku by měl být umístěn (pokud je to možné) takovým způsobem, že spolu s již umístěnými vrcholy třech prvků a jejich propojujícími čarami tvoří paralelogram. Výsledný obraz Booleovského podsvazu tvoří krychli stojící na jednom z jejích rohů. Teď již by neměl být problém rozpoznat druhý Booleovský podsvaz, sestávající z prvků 2, 3, 5, 9, 6, 11, 14 a 16. Jelikož reprezentující krychle sdílí prvky 2, 5, 9 a 14 s první krychlí, je nakreslení této druhé krychle nasnadě. Dále by jsme s pravidlem paralelogramů nevystačili. Musíme proto zavést další, *pravidlo čar*. Podle něj čára do vrcholu nového prvku má být pokračováním nějaké již existující čáry. Když užijeme pravidel čar a paralelogramů pro vrcholy zbývajících prvků 8, 10, 13 a 15, obdržíme z geometrického diagramu uspokojivý diagram, do kterého pouze doplníme vrchol nejmenšího prvku 18 (viz obrázek 20.)

Geometrický diagram slouží jako pomocník při tvorbě dobrých výsledných diagramů. Je výhodné sledovat geometrické vzory a jejich realizace v diagramech. V některých (relativně výjimečných) případech je lepší konstruovat diagram zdola nahoru od nejmenšího prvku.

Materiál pro popis této metody jsem čerpal z [4]. Podobný náhled na tuto geometrickou metodu lze nalézt též v [11]. Zde je uvedený i příklad konkrétního použití této metody.



Obrázek 20. Vzorový diagram vytvořený geometrickou metodou

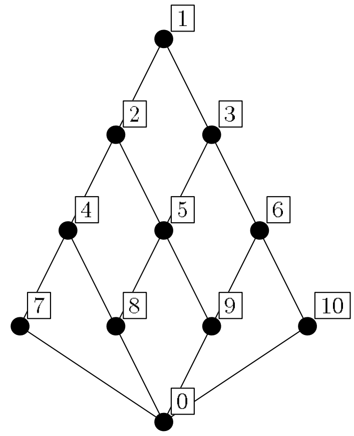
3.2.6. Hodnocení diagramu a metod

V této části se pokusím o porovnání výše rozebraných metod generování diagramu uspořádané množiny nebo svazu.

Jednoduchá úrovňová metoda nemá žádné teoretické základy, ale přesto pro jednoduché a malé množiny a svazy produkuje přijatelný diagram. Je to dáno tím, že množiny s malým počtem prvků (řekněme do 15) lze s výhodou kreslit po úrovních či vrstvách jednoduchým způsobem bez výrazného zhoršení čitelnosti diagramu. Navíc má metoda i jakousi heuristiku snížení počtu křížení hran, což jejím výsledkům jenom přidává na kvalitě. Pro větší množiny již heuristika nestačí a diagramy se rychle stávají téměř nečitelnými. Metoda není zaměřená na nějakou podmnožinu uspořádaných množin, dokáže generovat diagram libovolné uspořádané množiny. Podtrženo a sečteno, na to, že je tato metoda v podstatě amatérským pokusem o kreslení diagramu uspořádané množiny, diagramy jí produkované nejsou nejhorší a v určitých situacích se dají použít.

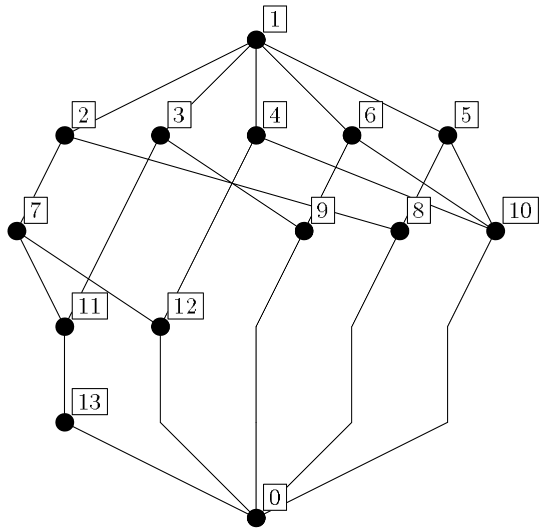
Příklad diagramu vytvořeného úrovňovou metodou je na obrázku 21.

Vrstvová metoda má naopak velmi hluboké a propracované základy, existuje spousta variant dílčích postupů v ní použitých a každý má různou úroveň úspěšnosti při postupném zpracovávání diagramu. Výsledky metody jsou oproti předchozí metodě obstojné i u větších množin a svazů. Je to hlavně díky účinným a teoreticky ověřeným a potvrzeným heuristikám redukce křížení hran diagramu. Velký podíl na kvalitě diagramů má také myšlenka nahrazení dlouhých hran lomenými hranami, což nemalou měrou zvyšuje čitelnost a přehlednost těchto diagramů. Metoda pracuje nad obecnými uspořádanými množinami, stejně jako předchozí metoda. Její důkladně propracované postupy vycházejí z obecnější teorie zobrazování grafů, které je věnováno mnoho knižních titulů a prací.



Obrázek 21. Příklad diagramu vytvořeného úrovňovou metodou

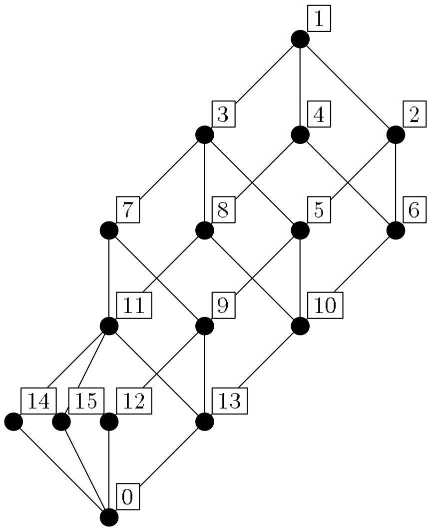
Příklad diagramu vytvořeného vrstvovou metodou je na obrázku 22.



Obrázek 22. Příklad diagramu vytvořeného vrstvovou metodou

Poslední prezentovaná metoda, geometrická, je podle mého subjektivního názoru nejlepší. Metoda výrazným způsobem využívá pravidelných geometrických konstrukcí v diagramu, jako např. rovnoběžné hrany, pravidelné n-rozměrné geometrické útvary (obdélníky a kvádry). Toto vyplývá se snahy sledovat strukturu svazu a vyhledávat v ní a optimálně zobrazovat standardní podstruktury, jako jsou Booleovy svazy. Ostatní prvky svazu, navazující na tyto struktury jsou umístěny vhodně tak, aby byla co nejvíce dodržena jakási geometrická pravidelnost celého diagramu. V této „geometrické interpretaci“ svazu spatřuji největší sílu metody a relativně vysokou dosaženou kvalitu diagramů. Na druhou stranu ale i když je používání paralelogramů velice výhodné, tato heuristika není založena na specifických strukturálních vlastnostech svazu, nýbrž na geometrických vlastnostech jeho podsvazů. Z předchozího lze vyvodit, že metoda je navržena pro generování diagramů svazů, ne obecných uspořádaných množin. Nicméně metoda si dokáže celkem přijatelně poradit i s těmito „obecně geometricky nepravidelnými“ strukturami.

Příklad diagramu vytvořeného geometrickou metodou je na obrázku 23.



Obrázek 23. Příklad diagramu vytvořeného geometrickou metodou

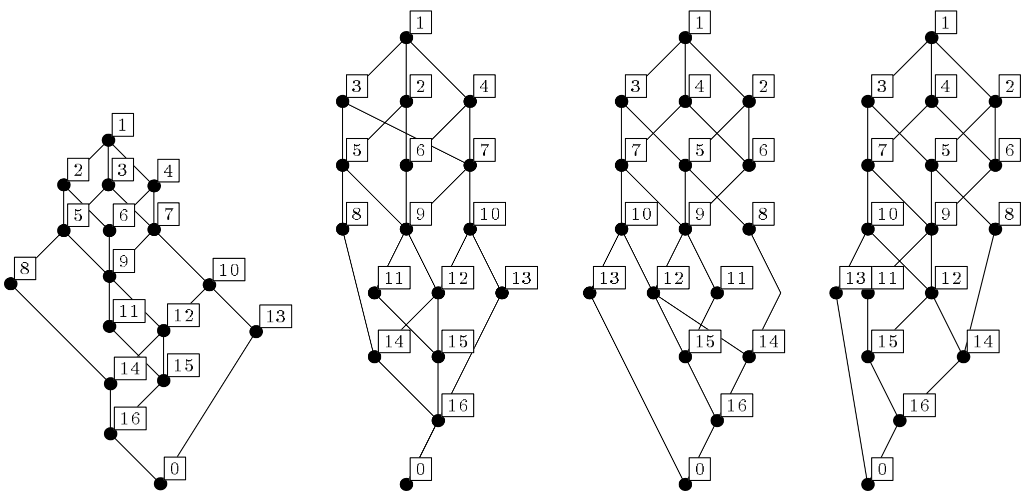
Pro názorné porovnání metod a kvality jimi vytvářených diagramů jsou určeny následují obrázky. Na obrázku 24. je první srovnávací sada vygenerovaných diagramů, zleva postupně ručně vytvořený, generovaný úrovňovou, vrstvovou a geometrickou metodou. Druhou sadu diagramů porovnávající jednotlivé metody můžete vidět (ve stejném pořadí) na obrázku 25.

Pro zajímavost dodávám ještě výkonnostní výsledky metod. Na obrázku 26. je zobrazen graf času potřebného pro vygenerování diagramu úrovňovou metodou pro uspořádanou množinu čítající N prvků. Na obrázcích 27. a 28. jsou odpovídající grafy pro vrstvovou a geometrickou metodu pro stejné množiny prvků. Obrázky nechám bez dalšího komentáře.

3.2.7. Souhrn a další metody

Metody generování diagramu uspořádaných množin a svazů prezentované v předchozích částech jsou jen malým zlomkem existujících metod, algoritmů a pravidel, používaných pro kreslení obecně grafů a rozebíraných v literatuře věnované kreslení grafů. Základní kroky těchto metod mohou být různými způsoby kombinovány, což nám dává mnohem větší počet různorodých metod a přístupů.

Na samotný závěr sekce uvedu ještě další možné metody generování diagramu uspořádaných množin a svazů, které nejsou ve výsledném programu implemento-



Obrázek 24. Srovnávací sada 1 vygenerovaných diagramů - ruční, úrovňovou, vrstvovou a geometrickou metodou

vány. První využívá speciální struktury prvků svazu, druhá je metoda umožňující interaktivní vytváření diagramu, třetí je zajímavou kombinací předchozí a vrstvové metody a poslední, nastiňující tzv. vnořené diagramy je používána u větších až velkých svazů.

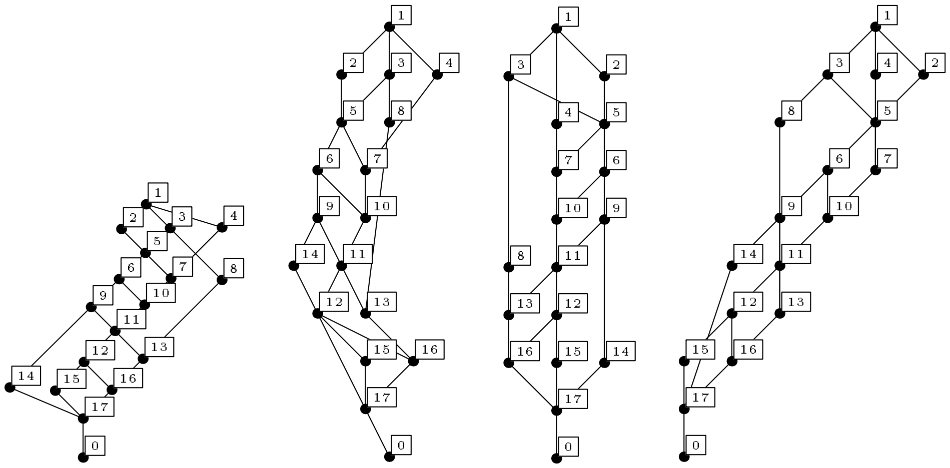
Z dalších metod, na které je možné narazit v nepřeberném množství různých titulů věnujících se problematice kreslení grafů jen jmenovitě např. topologická teorie grafů, geometrická teorie grafů, 3D zobrazování grafů, deklarativní přístupy, experimentální porovnávací studie, atd.

Mřížková metoda Je určena spíše pro speciální aplikace svazů jako např. *konceptuální svazy* používané v *konceptuální analýze dat*, kde mají prvky svazu určitou strukturu, kterou tato metoda využívá. Při konstrukci diagramu se využívají množiny atributů konceptů.

Tato metoda je založená na projekci svazu umístěného ve vícerozměrné mřížce do vhodné promítací roviny. Rozměr mřížky je určen minimálním počtem řetězců na které se dá rozložit množina atributů svazu. Vrcholy diagramu svazu se potom umístí na patřičná místa v této mřížce. Výsledný obrazec je nakonec promítnut do vhodně orientované roviny. Zásadním problémem této metody je právě volba této promítací roviny. Ve většině případů bývá nejlepší interaktivní naklánění promítací roviny, dokud není dosaženo akceptovatelného diagramu svazu.

Tato metoda se od ostatních podstatně liší. Nejdůležitější je, že výsledný diagram reflektuje strukturální analýzu svazu.

Hluboké teoretické základy této metody jsou rozebírány např. v publikaci [10] nebo v článcích [8] a [13].

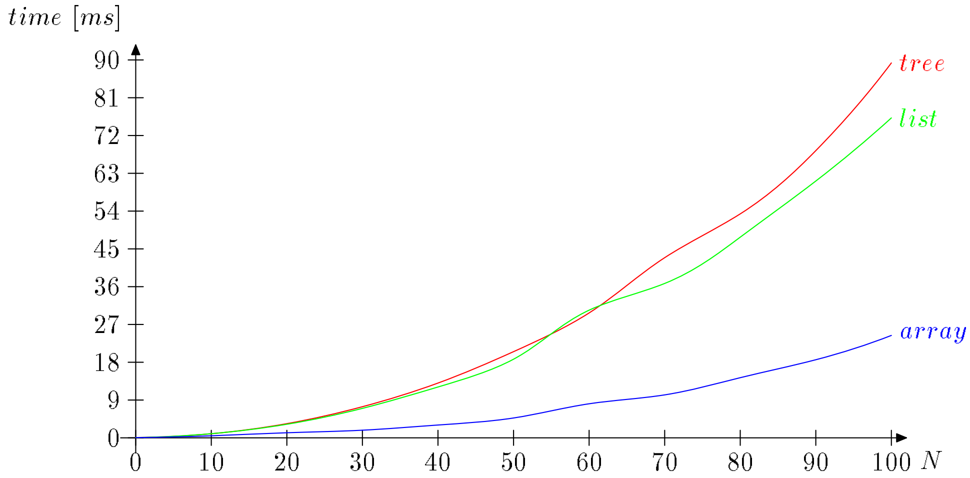


Obrázek 25. Srovnávací sada 2 vygenerovaných diagramů - ruční, úrovňovou, vrstvovou a geometrickou metodou

Aditivní metoda Tato metoda vytváří *aditivní diagram* uspořádané množiny. Metoda vychází z *pravidla umístění*, které přiřazuje prvkům uspořádané množiny body v rovině. Jestliže a a b jsou prvky uspořádané množiny, pro které platí $a < b$, pak podle tohoto pravidla musí být bod přiřazený prvku a níže než bod přiřazený prvku b. Dále se definují dvě zobrazení, *množinová reprezentace* a *mřížková projekce*. Množinová reprezentace rep uspořádané množiny P je zobrazení z této množiny do množiny podmnožin nějaké množiny X (např. opět původní uspořádané množiny P), $\mathrm{rep} : P \to \mathfrak{B}(X)$, s vlastností $x \leq y \Leftrightarrow \mathrm{rep}\, x \subseteq \mathrm{rep}\, y$, kde x a y jsou prvky z P. Mřížková projekce je zobrazení z množiny X do množiny $\mathbb{R}^2$, $\mathrm{vec} : X \to \mathbb{R}^2$, přiřazující každému prvku z množiny X reálný vektor s kladnou y-souřadnicí. Vektor pozice prvku p v diagramu dostaneme jako součet vektorů přiřazených všem prvkům z množiny $\mathrm{rep}\, p$ a přičtení libovolného vektoru posunutí, $\mathrm{pos}\, p := n + \sum_{x \in \mathrm{rep}\, p} \mathrm{vec}\, x$. Výsledný diagram je charakterizován velkým počtem rovnoběžných hran, což zvyšuje jeho čitelnost. Více lze k této metodě nalézt v [4] nebo [2].

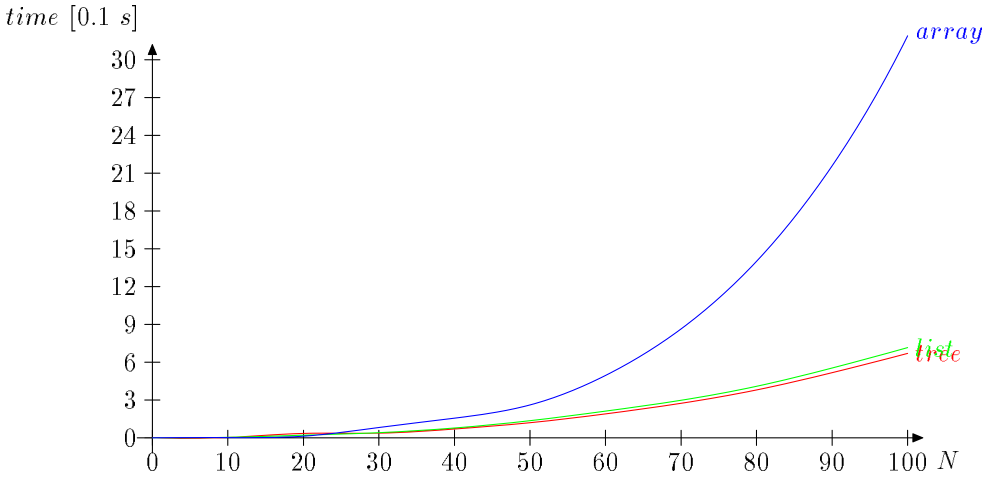
Hybridní metoda Tato metoda kombinuje myšlenky aditivní a vrstvové metody. Používá některé přiřazení do vrstev z vrstvové metody (často metodu nejdelší cesty) a aditivní metodu pro určení horizontálních souřadnic vrcholů. Metoda navíc řeší i problém průchodu nějaké hrany vrcholem. K tomu se používají postupy zpětného zpracování diagramu (*backtracking*), kdy se postupně hledá nejlepší diagram podle určitých kritérií.

Více k této metodě viz [2].



Obrázek 26. Čas vygenerování diagramu úrovňovou metodou pro množinu s N prvky

Vnořené (nested) diagramy Pro diagramy čítající 50 a více prvků se používají *vnořené (nested) diagramy*. Základní myšlenka vnořených diagramů spočívá v oddělení částí obyčejného diagramu a nahrazení svazků rovnoběžných čar mezi těmito částmi jednou čarou. Vnořený diagram tedy tvoří ohraničené obdélníky, které obsahují části (pod-diagramy) obyčejného diagramu a které jsou spojeny čarami. V případě propojení dvou obdélníků jednoduchou čarou tato nahrazuje rovnoběžné čáry propojující všechny prvky pod-diagramů obyčejného diagramu v obdélnících. Dvojitá čára mezi obdélníky znamená, že každý prvek pod-diagramu v horním obdélníku je větší než každý prvek pod-diagramu v dolním obdélníku. Vnořené diagramy mají ale ještě další možnosti, např. že pod-diagramy obyčejného diagramu v obdélnících propojených jednoduchou čarou nemusejí být stejné, ale mohou tvořit jen část stejných pod-diagramů. Více lze o vnořených diagramech nalézt opět v [4] nebo jejich důkladný teoretický rozbor v [13]. Na obrázku 29. je příklad obyčejného diagramu uspořádané množiny a na obrázku 30. je jemu odpovídající vnořený diagram téže uspořádané množiny. Velké uplatnění nacházejí vnořené diagramy v zobrazování konceptuálních svazů.



Obrázek 27. Čas vygenerování diagramu vrstvovou metodou pro množinu s N prvky

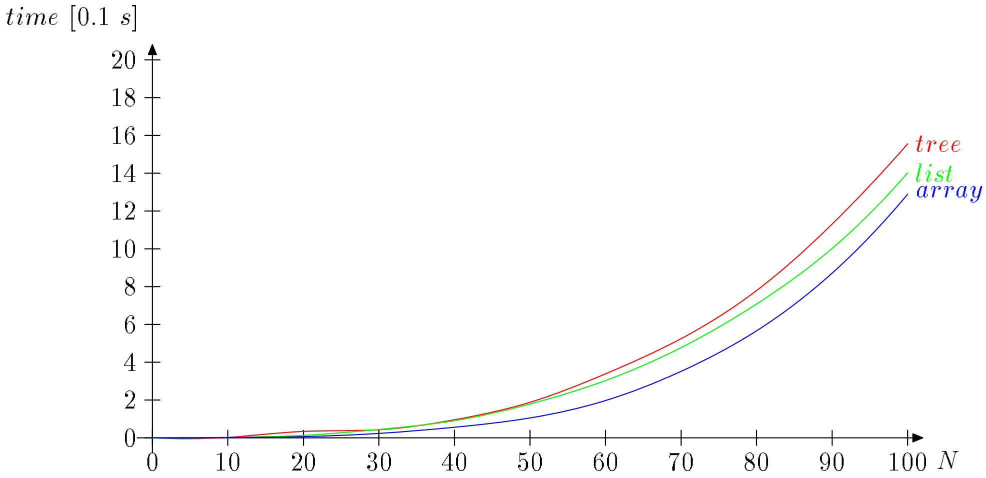
4. Popis programu

Program je napsán v jazyce C++ se snahou co nejvíce dodržet normu *ANSI C++*. Při designu a implementaci aplikace jsem se snažil dodržovat principy objektově orientovaného přístupu k návrhu programu a metody objektově orientovaného programování (*OOP*). V program je důsledně oddělena funkční část programu od jeho prezentace, tj. uživatelského rozhraní.

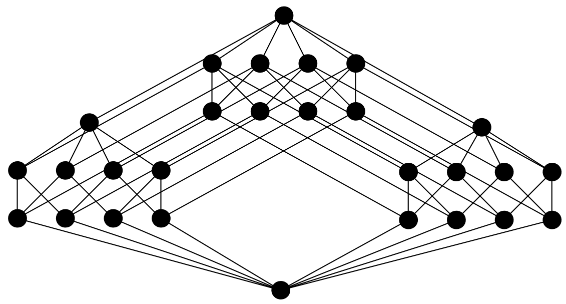
Program byl kompletně napsán pod operačním systémem Debian GNU/Linux (`http://www.debian.org/`) v editoru GNU Emacs (`http://www.gnu.org/software/emacs/`). Program je tedy primárně určen pro provozování pod tímto systémem a vůbec pod unixovými systémy. Protože jsou ale v počítačovém světě stále nejrozšířenější systémy Windows, byl program portován i na tyto systémy. Přitom byly zachovány jeho stejné vlastnosti, funkčnost i vzhled. Jediné rozdíly jsou ty, že pod systémem Windows se nevytváří samostatná knihovna jádra programu (viz dále) a že pod unixovými systémy má program kromě GUI i jednoduché volby příkazové řádky při spuštění programu.

Program je navíc plně internacionalizován. Původně je napsán v anglickém jazyce, ale je připraven pro bezproblémové přeložení do jiného jazyka. Jako ukázku této možnosti jsem program přeložil do mého rodného jazyka, češtiny. Pro spuštění programu v jiném jazyce jej není potřeba znovu překládat ze zdrojových souborů, stačí standardními postupy změnit jazykové a národní prostředí operačního systému (tzv. *lokalizaci*) a potom program jednoduše spustit znovu.

Základní instrukce pro instalaci a spuštění programu jsou uvedeny v souboru `README`. Pro systémy Windows jsou tyto informace v souboru `README.windows`.



Obrázek 28. Čas vygenerování diagramu geometrickou metodou pro množinu s N prvky



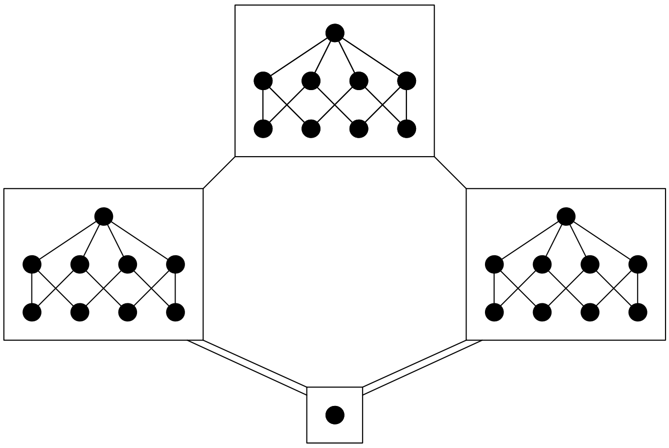
Obrázek 29. Obyčejný diagram odpovídající vnořenému diagramu na obrázku 30.

4.1. Požadavky

Na program se samozřejmě kladou určité požadavky, které by měl splňovat. Ne jinak tomu bylo u tohoto programu.

Mezi základní požadavky kladené na program patřily tyto:

- implementace a porovnání různých datových reprezentací uspořádané množiny a svazu
- implementace a porovnání různých metod generování Hasseova diagramu uspořádané množiny a svazu
- objektový přístup k implementaci a orientace na komponentové rozdělení programu
- podrobná dokumentace implementovaných datových reprezentací a metod generování diagramu, jejich srovnání



Obrázek 30. Vnořený (nested) diagram odpovídající diagramu na obrázku 29.

Jako další požadavky a cíle programu jsem si stanovil a v programu implementoval:

- interaktivní vytváření a editace uspořádané množiny i jejího diagramu, včetně neomezených operací vrácení a zrušení vrácení (undo-redo)
- možnost označení různých podmnožin prvků uspořádané množiny, které byly definovány v teoretické části 2.
- přijatelně výkonné vytváření diagramu množiny a vůbec práce s množinou
- efektivní grafické uživatelské rozhraní
- ukládání a načítání uspořádané množiny a jejího diagramu do XML souboru
- export diagramu do METAPOSTu
- internacionalizace, tj. umožnění více jazykových mutací programu
- portace na další platformy a operační systémy

Jak byly jednotlivé požadavky a vytyčené cíle zde vyjmenované splněny, můžete posoudit sami.

4.2. Komponentová struktura

V programu je důsledně oddělena základní funkční část programu od její grafické prezentace pomocí GUI. Funkční základ programu tvoří knihovna *liblatvis*, prezentovaná v následující podkapitole, zbytek programu potom v další podkapitole.

4.2.1. Knihovna liblatvis

Knihovna *liblatvis* představuje jádro, funkční základ programu. V knihovně jsou implementovány datové reprezentace množiny a metody generování diagramu. Její implementace je nezávislá na použité platformě a operačním systému. Knihovnu je možné používat samostatně nezávisle na zbytku programu. Pro použití ve spojení s jiným programem jsou určeny hlavičkové soubory definující rozhraní knihovny. V těchto souborech jsou uvedeny kompletní deklarace základních tříd, které lze v jiném programu vytvářet a používat. De facto příkladem takového použití knihovny v programu je samotná implementace GUI programu.

Hlavičkové soubory rozhraní knihovny jsou `element.h`, `diagram_graphic.h`, `orderedset.h`, `orderedsets.h` a `undoredo.h`.

Kód knihovny je dostatečně okomentován, hlavně implementace operací na uspořádané množině a nejvíce potom jednotlivé postupy a kroky metod generování diagramu.

Knihovna se nevytváří při překladu programu pod systémy Windows, jak bylo zmíněno výše. Pod tímto systémem je knihovna spojena se zbytkem programu do jednoho výsledného spustitelného souboru.

4.2.2. Aplikace LatVis

Aplikace *LatVis* sestává kromě knihovny *liblatvis* také z uživatelského rozhraní - GUI. Jednotlivé funkce uživatelského rozhraní samozřejmě využívají knihovnu a jí poskytnuté rozhraní. Toto uživatelské rozhraní je již silně závislé na operačním systému, pod kterým je program provozován.

Pod unixovými operačními systémy, speciálně pod operačním systémem GNU/Linux, existuje dvojí implementace GUI. První používá grafickou knihovnu gtkmm (`http://www.gtkmm.org/`), druhá grafickou knihovnu Qt (`http://www.trolltech.com`). Navenek vypadají obě verze GUI stejně, to byl záměr, viz dále. Existence dvojí implementace GUI má svůj důvod. Nejdříve jsem začal psát rozhraní pomocí knihovny gtkmm, protože je mi z různých osobních důvodů bližší a již jsem s ní měl nějaké zkušenosti z jiných programů. V průběhu práce na programu ale najednou vyvstal požadavek funkčnosti konkrétně pod operačním systémem Windows. A jelikož knihovna Qt je multiplatformní a existuje její implementace pro více operačních systémů, pro unixové i Windows, rozhodl jsem se použít ji. Navíc jsem postupem času zjistil, že tato knihovna je výborně navržená s ohledem právě na funkčnost na více operačních systémech.

Původního gtkmm GUI se mi ale nechtělo vůbec vzdát, tak jsem pokračoval v implementaci obojí verze, gtkmm i Qt. To mi navíc přineslo tu možnost, porovnat způsob programování v obou grafických knihovnách. Výsledek mého zjištění je ten, že v obou knihovnách se stejné věci provádějí v podstatě stejně.

Pod systémy Windows je implementováno pouze GUI založené na grafické knihovně Qt. gtkmm verzi se mi na tomto systému nepodařilo zprovoznit.

Vzhled GUI plně závisí na nastavení grafického vzhledu konkrétního operačního systému. Pro změnu tohoto vzhledu jsou určena tzv. *témata* uživatelského rozhraní systému (Windows) nebo grafické knihovny použité pro zobrazování oken (témata pro gtkmm a Qt knihovny pod unixovými systémy).

Pod unixovými systémy má program *LatVis* kromě grafického rozhraní i jednoduché volby příkazové řádky systému:

`-h --help`
: Vypíše stručnou nápovědu k tomuto použití programu.

`-v --version`
: Vypíše verzi spuštěného programu.

`-d --diagram`
: Volba instruuje program ke generování diagramu uspořádané množiny načtené ze souboru.

`-m --method` *metoda*
: Vybere metodu generování diagramu. Parametr volby *metoda* může nabývat hodnot: 1 pro volbu úrovňové metody, 2 pro geometrickou metodu (implicitní) a 3 pro volbu vrstvové metody generování diagramu.

`-s --save`
: Uloží načtenou množinu do stejného souboru, z jakého byla načtena. Při zadání této volby se nespouští GUI a spolu s množinou je do souboru uložen i její vygenerovaný diagram, pokud byla zadána volba `--diagram`.

Ostatní parametry zadané programu jsou považovány za jména souborů, ze kterých se mají uspořádané množiny načítat. Souborů je možné zadat více, program zpracuje postupně všechny jeden za druhým.

4.3. Základní třídy - programátorský popis

Základní funkční třídy programu jsou obsaženy v knihovně *liblatvis*. Jedná se o třídy datové reprezentace uspořádané množiny, třídy vytvořeného diagramu, třídu grafické podoby diagramu, třídu samotné uspořádané množiny a v neposlední řadě také o třídu spravující vytváření a zánik uspořádaných množin.

Kromě samotné uspořádané množiny se zde pracuje i s obyčejnými množinami prvků, které jsou realizovány pomocí abstraktního datového typu množiny implementovaného v šabloně množiny *set* z balíku šablon standardních datových struktur a základních univerzálních algoritmů, STL (`http://www.sgi.com/tech/stl/`). Pro seznam se používá šablona *list* z tohoto balíku šablon, která implementuje obousměrný seznam.

4.3.1. Datová reprezentace

V podsekci 3.1. byly popsány tři možné datové reprezentace uspořádané množiny. Každou tuto reprezentaci realizuje v knihovně *liblatvis* jedna třída. Datovou reprezentaci pomocí dvourozměrného pole realizuje třída *OrderData_Array*, reprezentaci pomocí seznamu seznamů pak třída *OrderData_List* a nakonec reprezentaci pomocí binárního vyhledávacího stromu realizuje třída *OrderData_Tree*. Všechny tyto třídy mají stejné rozhraní, liší se pouze svojí implementací a uložením dat uspořádané množiny. Jsou proto všechny zděděny ze společného předka, třídy *OrderData*, který definuje toto rozhraní.

Prvek uspořádané množiny, se kterým třídy pracují jako se základní datovou jednotkou, představuje typ *ElementId* (definovaný v souboru `element.h`), což je přejmenovaný jednoduchý celočíselný datový typ. Takto jednoduchý datový typ jsem volil proto, protože je podle mě pro daný účel plně dostačující. Na této úrovni implementace uspořádané množiny požadujeme od jejích prvků pouze rozlišitelnost jednoho od druhého a na to obyčejné číslo stačí. Konstanta *NoElement* je speciální hodnotou tohoto typu, představující nulový prvek uspořádané množiny. Konstanta funguje podobně jako hodnota *NULL* v jazyce C++.

Nyní již následují programátorské popisy jednotlivých tříd.

class OrderData;

Abstraktní třída reprezentace dat uspořádané množiny definující rozhraní této reprezentace dat, deklarována v souboru `orderdata.h`.

Důležité metody třídy:

void clear();

vymaže obsah množiny, tj. veškerá data

bool newElement(ElementId x);

vloží do množiny nový prvek x, pokud ovšem v množině již takový prvek neexistuje bool removeElement(ElementId x);

vyjme prvek x z množiny, s tímto prvkem se z množiny odstraní také všechny relační vazby na tento prvek

bool orderElements(ElementId x, ElementId y);

vytvoří mezi prvky x a y v množině relační vazbu $x < y$, vazba se nevytvoří, pokud již v množině mezi těmito prvky existuje vazba opačná, tj. $y < x$, automaticky se vytváří indukované tranzitivní vazby

bool removeOrder(ElementId x, ElementId y);

odstraní z množiny relační vazbu mezi prvky x a y, indukovanou tranzitivní vazbu mezi těmito prvky nelze odstranit

bool less(ElementId x, ElementId y, bool $transitivity$);

vrací booleovskou hodnotu značící, zda platí vztah $x < y$, parametr $transitivity$ určuje, zda se má jednat o prvky předchůdce a následovníka, nebo jestli se má uvažovat i tranzitivní vztah těchto prvků

void getElements(set<ElementId> &s);

v parametru s vrací množinu všech prvků uspořádané množiny

void upperElements(set<ElementId> &s, ElementId x);

v parametru s vrací množinu všech následovníků prvku x

void lowerElements(set<ElementId> &s, ElementId x);

analogie předchozí metody pro předchůdce

void minimalElements(set<ElementId> &s, set<ElementId> ss);

v parametru s vrací množinu všech minimálních prvků z podmnožiny ss prvků

void maximalElements(set<ElementId> &s, set<ElementId> ss);

analogie předchozí metody pro maximální prvky

ElementId leastElement(set<ElementId> ss);

vrací nejmenší prvek z podmnožiny ss prvků, nebo *NoElement*, pokud takový prvek v uspořádané množině neexistuje

ElementId greatestElement(set<ElementId> ss);

analogie předchozí metody pro největší prvek

void lowerCone(set<ElementId> &s, set<ElementId> ss);

v parametru s vrací množinu dolní kužel podmnožiny ss prvků

void upperCone(set<ElementId> &s, set<ElementId> ss);

analogie předchozí metody pro horní kužel

ElementId infimum(set<ElementId> ss);
: vrací infimum podmnožiny ss prvků, nebo *NoElement*, pokud takový prvek v uspořádané množině neexistuje

ElementId supremum(set<ElementId> ss);
: analogie předchozí metody pro supremum

void getPaths(list<set<ElementId> > &l, ElementId x, ElementId y);
: v parametru l vrací seznam všech cest v uspořádané množině z prvku x do prvku y

class OrderData_Array : public OrderData;
: Třída reprezentace dat uspořádané množiny pomocí dvourozměrného pole, implementuje rozhraní ze zděděné třídy *OrderData*, implementována v souborech `orderdata_array.h` a `orderdata_array.cc`.

Důležitá data třídy:

ElementId *elements;
: jednoduché pole s prvky uspořádané množiny

table_t **table;
: dvourozměrné pole představující tabulku relace uspořádání množiny, v tabulce je v příslušném políčku číslo 1, když jsou prvky x a y spolu v relaci $x \leq y$, jinak číslo 0, typ *table_t* je jednoduchý celočíselný typ

class OrderData_List : public OrderData;
: Třída reprezentace dat uspořádané množiny pomocí seznamu seznamů, implementuje rozhraní ze zděděné třídy *OrderData*, implementována v souborech `orderdata_list.h` a `orderdata_list.cc`.

Důležitá data třídy:

set<ListItem> list_upper;
: seznam seznamů následovníků pro každý prvek množiny, seznamy jsou implementovány pomocí STL šablony *set* pro rychlejší přístup ke konkrétnímu seznamu nebo prvku seznamu, *ListItem* je struktura položky seznamu, obsahující prvek uspořádané množiny a seznam následovníků tohoto prvku

set<ListItem> list_lower;
: seznam seznamů předchůdců pro každý prvek množiny, analogicky k předchozímu seznamu, třída obsahuje oba dva seznamy, protože pro některé implementace operací uspořádané množiny je lepší použití seznamů následovníků, pro jiné zase seznamů předchůdců

class OrderData_Tree : public OrderData;

Třída reprezentace dat uspořádané množiny pomocí Red-Black binárního vyhledávacího stromu, implementuje rozhraní ze zděděné třídy *OrderData*, implementována v souborech `orderdata_tree.h` a `orderdata_tree.cc`.

Důležitá data třídy:

set<Pair *> pairs_first;

Red-Black strom obsahující uspořádané dvojice $\langle x, y \rangle$ relace pokrytí uspořádané množiny setříděný podle první položky dvojice, dvojice je implementována pomocí struktury *Pair* obsahující první a druhý prvek dvojice, samotné stromy jsou implementovány pomocí STL šablony *set*, protože tato šablona implementuje množinu prvků právě pomocí Red-Black stromu

set<Pair *> pairs_second;

Red-Black strom obsahující uspořádané dvojice $\langle x, y \rangle$ relace pokrytí uspořádané množiny setříděný podle druhé položky dvojice, analogicky k předchozímu stromu, třída obsahuje oba dva stromy ze stejného důvodu, jako dva seznamy u třídy *OrderData_List*, tento i předchozí strom obsahují jako uzly pouze ukazatele na struktury dvojic, protože oba stromy používají stejné dvojice

4.3.2. Vytvořený diagram

Různé metody generování diagramu byly podrobně popsány v podsekci 3.2. Každá tato metoda je implementována ve své třídě. Třída realizující generování diagramu jednoduchou úrovňovou metodou se jmenuje *LevelDiagram*, vrstvovou metodu realizuje třída *LayerDiagram* a nakonec třída realizující geometrickou metodu je pojmenována *GeometricDiagram*. Všechny tyto třídy mají opět stejné rozhraní, liší se pouze svojí implementací tohoto rozhraní a tedy implementovanou metodou generování diagramu. Všechny jsou zděděny ze společného předka, třídy *Diagram*, který definuje toto rozhraní.

Metody těchto tříd generují diagram sestávající z vrcholů, které jsou implementovány třídou *Vertex_e*.

Následují programátorské popisy jednotlivých tříd.

class Vertex_e;

Třída implementující vrchol generovaného diagramu, implementována v souboru `diagram.h`.

Důležitá data třídy:

ElementId element

prvek uspořádané množiny, ke kterému vrchol patří

ElementId dummy

prvek odpovídající vrcholu z nějakého řetězce falešných vrcholů, používaných u vrstvové metody

Coord_e_t x

x-souřadnice vrcholu ve vygenerovaném diagramu, typ *Coord_e_t* je jednoduchý celočíselný typ

Coord_e_t y

y-souřadnice vrcholu ve vygenerovaném diagramu

class Diagram;

Abstraktní třída generovaného diagramu definující rozhraní generovaného diagramu, deklarována v souboru `diagram.h`.

Důležité metody třídy:

void generateDiagram(generationParams_t *$params$);

hlavní metoda generující diagram uspořádané množiny, v parametru $params$ jsou uloženy různé parametry generování, které jsou specifické pro každou metodu, proto je typ definován až v dědicích třídách implementujících konkrétní metody

void getDiagram(set<Vertex_e> &s);

v parametru s vrací vrcholy vygenerovaného diagramu, hrany jsou zjištěny ze samotné uspořádané množiny

class LevelDiagram : public Diagram;

Třída implementující generování diagramu úrovňovou metodou, implementuje rozhraní ze zděděné třídy *Diagram*, implementována v souborech `leveldiagram.h` a `leveldiagram.cc`.

Důležitá data třídy:

OrderData *data;

data uspořádané množiny, pro kterou se má diagram generovat

set<Vertex_e> verteces;

množina vrcholů vygenerovaného diagramu

generationParams_t *generationParams;

parametry generování diagramu, typ *generationParams_t* je struktura obsahující pouze jeden parametr, a to zda se má diagram generovat směrem od minimálních prvků nebo směrem od maximálních

class LayerDiagram : public Diagram;

Třída implementující generování diagramu vrstvovou metodou, implementuje rozhraní ze zděděné třídy *Diagram*, implementována v souborech `layerdiagram.h` a `layerdiagram.cc`.

Důležitá data třídy:

OrderData *data;
: data uspořádané množiny, pro kterou se má diagram generovat

set<Vertex_e> verteces;
: množina vrcholů vygenerovaného diagramu

OrderData *dummy_chains_data;
: uspořádaná množina prvků vrcholů z řetězců falešných vrcholů

list<list<Vertex_e *> *> layers;
: seznam vrstev diagramu, každá vrstva je seznam vrcholů

crossing_number_t **crossing_numbers;
: dvourozměrné pole (tabulka) čísel křížení hran mezi dvěma právě zpracovávanými vrstvami diagramu při redukci křížení, typ *crossing_number_t* je jednoduchý celočíselný typ

generationParams_t *generationParams;
: parametry generování diagramu, typ *generationParams_t* je struktura obsahující parametry, zda se má diagram generovat směrem od maximálních prvků nebo směrem od minimálních, která metoda rozvržení do vrstev se má použít (metoda nejdelší cesty nebo Coffman-Grahamův algoritmus), číslo (typu *cg_width_t*, což je jednoduchý celočíselný typ) udávající maximální šířku Coffman-Grahamova rozvržení do vrstev, jaká reprezentace dat uspořádané množiny se má použít pro množinu prvků falešných vrcholů a která metoda redukce křížení se má aplikovat (třídění prohozením sousedů, třídění rozdělením, metoda průměru nebo metoda mediánu)

Důležité metody třídy:

OrderData *getDummyChainsData();
: vrací uspořádanou množinu prvků falešných vrcholů

void longestPathLayering();
: implementuje rozvržení do vrstev metodou nejdelší cesty

void CoffmanGrahamLayering(cg_width_t *width*);
: implementuje rozvržení do vrstev Coffman-Grahamovou metodou, s maximální šířkou *width* diagramu

void insertDummyVerteces();
: vkládá do diagramu cesty z falešných vrcholů

void countCrossingNumbers(list<Vertex_e *> *$l1$, list<Vertex_e *> *$l2$);
: počítá čísla křížení hran mezi vrstvami $l1$ a $l2$ diagramu a ukládá je do pole *crossing_numbers*

void adjacentExchangeCrossingReduction(list<Vertex_e *> *$layer$);
: implementuje redukci křížení metodou třídění prohozením sousedů, třídí vrcholy ve vrstvě *layer*

void splitCrossingReduction(list<Vertex_e *> *$layer$);
: implementuje redukci křížení metodou třídění rozdělením, třídí vrcholy ve vrstvě *layer*

void barycenterCrossingReduction(list<Vertex_e *> *$l1$, list<Vertex_e *> *$l2$);
: implementuje redukci křížení metodou průměru, třídí vrcholy ve vrstvě $l2$, sousední vrcholy jsou ve vrstvě $l1$

void medianCrossingReduction(list<Vertex_e *> *$l1$, list<Vertex_e *> *$l2$);
: implementuje redukci křížení metodou mediánu, třídí vrcholy ve vrstvě $l2$, sousední vrcholy jsou ve vrstvě $l1$

class GeometricDiagram : public Diagram;
: Třída implementující generování diagramu geometrickou metodou, implementuje rozhraní ze zděděné třídy *Diagram*, implementována v souborech `geometricdiagram.h` a `geometricdiagram.cc`.

Důležitá data třídy:

OrderData *data;
: data uspořádané množiny, pro kterou se má diagram generovat

set<Vertex_e> verteces;
: množina vrcholů vygenerovaného diagramu

generationParams_t *generationParams;
: parametry generování diagramu, typ *generationParams_t* je struktura obsahující pouze jeden parametr, a to zda se má diagram generovat směrem od největšího prvku nebo směrem od nejmenšího

Důležité metody třídy:

void placeVertexRuleOfParallelograms(Vertex_e &$vertex$);
: umístí vrchol *vertex* v diagramu podle pravidla paralelogramů

void placeVertexRuleOfLines(Vertex_e &$vertex$);
: umístí vrchol *vertex* v diagramu podle pravidla čar

4.3.3. Grafická podoba diagramu

„Grafickou podobu diagramu“ zastřešuje třída *Diagram_Graphic*. V této třídě je uložený zobrazený diagram. Rozhraní této třídy je zpřístupněno zvenku knihovny *liblatvis*.

Vrcholy diagramu jsou realizovány třídou *Vertex*, jejíž data a rozhraní jsou přístupné zvenčí knihovny *liblatvis*. Třída *Diagram_Graphic* definuje také dvě metody, používající třídu prvku uspořádané množiny, která se jmenuje *Element*. Tato třída je popsána v následující části popisující třídy uspořádané množiny.

class Vertex;
: Třída implementující vrchol diagramu uspořádané množiny, třída má stejný obsah jako třída *Vertex_e*, pouze pro souřadnice vrcholu se používá jednoduchý celočíselný typ *Coord_t*, implementována v souboru `element.h`

class Diagram_Graphic;
: Třída diagramu uspořádané množiny, implementována v souborech `diagram_graphic.h` a `diagram_graphic.cc`.

Důležitá data třídy:

OrderedSet *orderedset;
: uspořádaná množina, jejíž diagram třída obsahuje, třída *OrderedSet* je popsána v následující části

OrderData *data;
: data uspořádané množiny, jejíž diagram třída obsahuje a pro kterou se má generovat diagram

set<Vertex> verteces;
: množina vrcholů diagramu

OrderData *dummy_chains_data;
: uspořádaná množina prvků vrcholů z řetězců falešných vrcholů

Coord_t min_x, max_x, min_y, max_y;
: souřadnice obdélníku ohraničujícího diagram, tj. minimální a maximální souřadnice vrcholů v diagramu

generationParams_t generationParams;
: soubor všech parametrů generování diagramu pro všechny metody implementované ve třídách zděděných z třídy *Diagram*, typ *generationParams_t* je struktura obsahující všechny parametry ze zmíněných tříd, plus navíc jeden parametr určující metodu generování diagramu (úrovňová, vrstvová a geometrická)

Důležité metody třídy:

generationParams_t getGenerationParams();
: vrací parametry generování diagramu

bool generateDiagram(generationParams_t *params*);
: metoda generující diagram uspořádané množiny podle parametrů zadaných v *params*

Vertex *addVertex(Vertex v);
: přidá vrchol v do diagramu

bool deleteVertex(Vertex v);
: odstraní vrchol v z diagramu

Vertex *getVertex(Element x);
: vrací vrchol pro prvek x množiny

void getVerteces(set<Vertex> &s);
: v parametru s vrací všechny vrcholy diagramu

void clearDiagram();
: smaže diagram

Element *getElement(Vertex v);
: vrací prvek množiny pro vrchol v diagramu

bool isEdge(Vertex vx, Vertex vy);
: vrací *true*, pokud mezi vrcholy vx a vy je v diagramu hrana, jinak *false*

4.3.4. Uspořádaná množina

Uspořádanou množinu představuje třída *OrderedSet*. Rozhraní této třídy je zpřístupněno zvenku knihovny.

Prvky uspořádané množiny jsou realizovány pomocí třídy *Element*, jejíž data a rozhraní jsou též přístupné zvenčí knihovny *liblatvis*. Pro jméno prvku se používá STL šablona *string*.

Instance této třídy jsou vytvářeny a rušeny pomocí třídy *OrderedSets*, jejíž rozhraní je samozřejmě také zpřístupněno zvenčí knihovny. V této třídě jsou uspořádané množiny identifikovány pomocí identifikátoru *OrderedSetId*, což je jednoduchý celočíselný typ. Konstanta *NoOrderedSet* je speciální hodnotou tohoto typu, představující nulovou uspořádanou množinu. Konstanta funguje podobně jako hodnota *NULL* v jazyce C++.

Změny stavu této třídy jsou evidovány v seznamu undo_list. Tento seznam je implementován pomocí šablony *UndoRedo*.

class Element;

Třída prvku uspořádané množiny, implementována v souboru `element.h`.

Důležitá data třídy:

ElementId id;

prvek datové reprezentace uspořádané množiny

ElementName name;

jméno prvku uspořádané množiny, typ *ElementName* je STL šablona *string*

class OrderedSet;

Třída uspořádané množiny, implementována v souborech `orderedset.h` a `orderedset.cc`.

Důležitá data třídy:

UndoRedo<UndoRedoType *> undo_list;

seznam stavů uspořádané množiny, typ *UndoRedoType* je struktura, obsahující položky: všechny prvky množiny, třída reprezentace dat množiny, diagram a další informace o množině

OrderedSetId id;

identifikátor množiny pro použití ve třídě *OrderedSets*

set<Element> elements;

množina všech prvků uspořádané množiny

OrderData *data;

objekt třídy reprezentace dat množiny

data_t datatype;

zvolená reprezentace dat uspořádané množiny, typ *data_t* je výčtový typ, specifikující reprezentaci dat (dvojrozměrné pole, seznam seznamů nebo vyhledávací strom)

Diagram_Graphic *diagram;

diagram uspořádané množiny

Důležité metody třídy:

void clear();

vymaže množinu

void save_UndoRedo();

uloží stav uspořádané množiny do seznamu undo_list

void removeLast_UndoRedo();
: vyjme poslední uložený stav uspořádané množiny ze seznamu undo_list

void undo();
: provádí operaci undo, tj. vrátí stav uspořádané množiny do předchozího stavu

void redo();
: provádí operaci redo, tj. změní stav množiny na následující v seznamu undo_list

data_t getDataType();
: vrací typ reprezentace dat množiny

Element *newElement(ElementName x, Vertex *v = NULL);
: vloží do množiny nový prvek se jménem x a případně i s vrcholem v do diagramu

bool rename(Element x, ElementName $name$);
: změní jméno prvku x na $name$

bool removeElement(Element x);
: odstraní prvek x z množiny

bool orderElements(Element x, Element y);
: uspořádá prvky x a y množiny tak, že bude $x < y$

bool removeOrder(Element x, Element y);
: zruší uspořádání mezi prvky x a y v množině

bool less(Element x, Element y, bool $transitivity$ = true);
: vrací $true$, pokud mezi prvky x a y platí vztah $x < y$, parametr $transitivity$ udává, zda se mají uvažovat tranzitivní vztahy

void getElements(set<Element> &s);
: v parametru s vrací množinu všech prvků uspořádané množiny

void upperElements(set<Element> &s, Element x);
: v parametru s vrací množinu všech následovníků prvku x

void lowerElements(set<Element> &s, Element x);
: analogie předchozí metody pro předchůdce

void minimalElements(set<Element> &s, set<Element> ss);
: v parametru s vrací množinu všech minimálních prvků z podmnožiny ss prvků

void maximalElements(set<Element> &s, set<Element> ss);
: analogie předchozí metody pro maximální prvky

Element *leastElement(set<Element> ss);
: vrací nejmenší prvek z podmnožiny ss prvků, nebo $NULL$, pokud takový prvek v uspořádané množině neexistuje

Element *greatestElement(set<Element> ss);
: analogie předchozí metody pro největší prvek

void lowerCone(set<Element> &s, set<Element> ss);
: v parametru s vrací množinu dolní kužel podmnožiny ss prvků

void upperCone(set<Element> &s, set<Element> ss);
: analogie předchozí metody pro horní kužel

Element *infimum(set<Element> ss);
: vrací infimum podmnožiny ss prvků, nebo $NULL$, pokud takový prvek v uspořádané množině neexistuje

Element *supremum(set<Element> ss);
: analogie předchozí metody pro supremum

void getPaths(list<set<Element> > &l, Element x, Element y);
: v parametru l vrací seznam všech cest v uspořádané množině z prvku x do prvku y

Element *getElement(ElementName x);
: vrací prvek množiny se jménem x

Diagram_Graphic *getDiagram();
: vrací diagram uspořádané množiny

class OrderedSets;
: Třída kolekce uspořádaných množin, vytváří a ruší objekty třídy *OrderedSet*, v knihovně existuje pouze jedna instance této třídy, implementována v souborech `orderedsets.h` a `orderedsets.cc`.

Důležitá data třídy:

set<RefOrderedSet_t> sets;
: množina evidovaných uspořádaných množin, typ *RefOrderedSet_t* je struktura, jejíž položky jsou uspořádaná množina a její počet použití (reference counter)

Důležité metody třídy:

OrderedSet *newSet(OrderedSet::data_t *data_type*);
: vytvoří a vrací novou uspořádanou množinu s typem reprezentace dat podle *data_type*

bool refSet(OrderedSet *s);
: zvýší počet použití množiny s

bool unrefSet(OrderedSet *s);
: sníží počet použití množiny s, jakmile tento počet dosáhne hodnoty 0, smaže množinu

template<class T> class UndoRedo;
: Šablona třídy realizující seznam objektů typu T s metodami pro podporu operací vrácení a zrušení vrácení (undo a redo), implementována v souboru `undoredo.h`.

Důležitá data třídy:

Item *actual;
: aktuální prvek seznamu, typ *Item* je struktura obsahující položky hodnoty ukládaného datového typu a ukazatele na předchozí a následující prvek seznamu (jde tedy o obousměrný seznam)

Důležité metody třídy:

void save(T d);
: smaže všechny prvky za aktuálním prvkem (včetně) a uloží d na konec seznamu

void removeLast();
: odstraní poslední prvek seznamu, pokud je aktuální prvek $NULL$

T undo(T d);
: uloží d na konec seznamu, pokud je aktuální prvek $NULL$ a nastaví aktuální prvek na konec seznamu (čili právě vložený prvek), vrací prvek před aktuálním prvkem (a aktualizuje aktuální prvek) nebo d, pokud je aktuální prvek prvním v seznamu

T redo(T d);
: vrací prvek následující za aktuálním (a aktualizuje aktuální prvek), nebo d, pokud je aktuální prvek poslední v seznamu, smaže aktuální prvek, pokud je poslední v seznamu

4.4. GUI - uživatelský popis

Grafické uživatelské rozhraní programu (*GUI - Graphical User Interface*) je rozhraní programu, přes které uživatel program ovládá. U mého programu jej tvoří dva druhy oken. Jedno jediné *hlavní okno* programu a libovolný počet *oken diagramu*, v nichž je zobrazen diagram uspořádané množiny. Jelikož oken diagramu může být v danou chvíli libovolný počet, lze pracovat najednou s několika diagramy a několika uspořádanými množinami, nezávisle jednoho na druhém. Při následujícím popisu oken, dialogů, menu, tlačítek a dalších grafických prvků použitých v programu budu používat jejich implicitní anglická pojmenování a texty.

4.4.1. Hlavní okno

Obrázek 31. Hlavní okno programu LatVis

Po spuštění programu LatVis se na obrazovce objeví hlavní okno programu (viz obrázek 31.). V titulku okna je jméno programu - *LatVis*. Toto okno sestává z pruhu hlavního menu a sady tlačítek. V hlavním menu jsou pouze položky nezávislé na tom, jestli máme otevřenou nějakou uspořádanou množinu nebo ne. Tlačítka umožňují rychlejší přístup k nejpoužívanějším funkcím programu než pomocí popup menu z okna diagramu. Okno je možné zvětšovat (a na určitou minimální velikost také zmenšovat), přitom se zvětšují i všechna tlačítka. Uzavření tohoto okna zavře i všechna okna diagramu a ukončí aplikaci. Pokud budou při požadavku na uzavření okna nějaké diagramy a množiny neuložené (změněné), program ze uživatele zeptá, zda chce skutečně aplikaci ukončit a ztratit tak změny provedené v diagramech a množinách.

Téměř všechny položky menu mají přiřazeny klávesové zkratky (*shortcuts*), u nichž jsem se snažil dodržel „standardní“ zažité klávesy. Položky, jejichž jméno následují tři tečky, vedou k vyvolání nějakého dialogu. Každé podmenu je možné „vytrhnout“ do samostatného okna pomocí zvolení přerušované čáry nahoře rámečku každého podmenu. Tím se podmenu zobrazí v samostatném okně pro rychlejší přístup k jeho položkám. Toto okno menu lze zavřít jako každé jiné okno, ale ve verzi gtkmm také pomocí přerušované čáry se šipkou nahoře okna. Tato přerušovaná čára se šipkou (v Qt verzi bez šipky) je také v původním menu v programu, které zůstává na svém místě a lze tak používat obě totožná menu nezávisle na sobě.

Tlačítka můžeme považovat za jakýsi *toolbar*, známý z jiných aplikací. Slouží, podobně jako toolbar, k rychlejšímu přístupu k nejpoužívanějším funkcím programu. Tlačítka fungují jako přepínače, po jeho stisknutí levým tlačítkem myši toto zůstane jediné stisknuté a znamená to, že se aplikace „přepla do módu“ funkce tohoto tlačítka. Každému tlačítku je též přiřazena klávesová zkratka, podobně jako u menu. Tlačítka slouží jako rychlejší varianta ekvivalentních položek popup menu vyvolávaného z okna diagramu. Proto funkce, které zapínají, popíšu až u popisu tohoto popup menu. Zvolení položky popup menu má stejný efekt jako stlačení tlačítka. Navíc stisknuté tlačítko plní užitečnou funkci indikace, který mód programu je právě aktivní.

Hlavní menu:

File [ALT+F] ([ALT+PODTRŽENÉ PÍSMENO])
: v tomto menu jsou standardní položky vytvoření nové množiny, otevření existující množiny uložené v souboru a ukončení programu

- **New...** [CTRL+N]
  započne vytváření nové (prázdné) uspořádané množiny, zobrazí se dialog *New set*
- **Open...** [CTRL+O]
  otevření uložené množiny (a diagramu), vyvolá standardní systémový dialog otevření souboru, nazvaný *Load set.*
- **Quit** [CTRL+Q]
  ukončí program, má stejnou funkci jako standardní zavírací tlačítko okna a provádějí se proto i stejné činnosti jako při zavření hlavního okna

Help [ALT+H] ([ALT+PODTRŽENÉ PÍSMENO])
: standardní položka nápovědy, dialogu o programu atd.

- **About...**
  vyvolá dialog o programu, nazvaný *About the LatVis*

Dialogy spouštěné z položek menu:

New set
: Dialog vytvoření nové množiny, vyvolán z menu *File→New...* a je nemodální.

Lze v něm navolit datovou reprezentaci množiny (rámec *Internal data representation* ze tří možností, dvourozměrné pole (*Two-dimensional array*), seznam seznamů (*List of lists*) a Red-Black strom (*Red-Black tree*), přednastavené je pole. Dále je pak možno ještě zadat minimální velikost diagramu (*Minimal diagram size*) pomocí editačních políček pro šířku (*Width*) a výšku (*Height*), do kterých se zadává kladné číslo udávající rozměr, přednastavené hodnoty jsou 200 v obou směrech.

Dialog nelze potvrdit při zadání nekladného čísla do libovolného z políček velikosti diagramu. Tlačítkem *Reset* se hodnoty vrátí na přednastavené. Po potvrzení dialogu se zobrazí prázdné okno diagramu nové uspořádané množiny, viz další část.

Load set

Standardní systémový dialog otevření souboru, vyvolán z menu *File→Open*, nemodální.

V gtkmm verzi v něm lze zvolit typ souboru pomocí přepínače *File type* s těmito možnostmi: typ určený příponou souboru (*By Extension*) a nativní typ souboru programu (*LAT (LatVis)*), soubory mají standardní příponu *.lat*.

V Qt verzi je nadefinován souborový filtr. V něm lze zvolit možnosti: všechny soubory (*All files (*)*), v tomto případě se typ souboru určí z jeho přípony, nebo nativní typ souboru programu (*LAT (LatVis) files (*.lat)*).

Dialog lze potvrdit až při zadání nebo zvolení jména souboru, a to i klávesou ENTER, když je fokus na editačním políčku jména souboru. Po potvrzení dialogu se otevře množina (popřípadě i s diagramem, pokud je uložen) v okně diagramu. Pokud se během otevírání souboru vyskytne nějaká chyba, je zobrazen (modální) dialog s hláškou vysvětlující nastalou chybu.

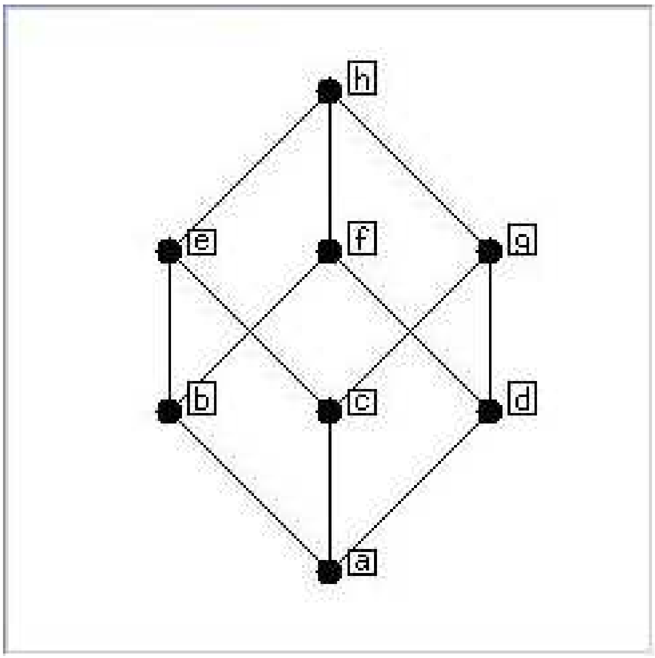
About the LatVis

Standardní dialog o programu, vyvolán z menu *Help→About*, nemodální.

Obsahuje informace o jménu a této verzi programu, navíc jméno autora a kontakt (email) na něj.

4.4.2. Okno diagramu

Okno diagramu uspořádané množiny zobrazuje vlastní diagram (viz obrázek 32.). V titulku okna je zapsáno jméno souboru množiny, pokud je množina nebo její diagram změněn, je před jménem v titulku hvězdička. Při zmenšování okna se automaticky objeví posuvné lišty (*scrollbars*), pokud se do okna nevleze celý diagram. Pomocí nich lze posouvat výřez zobrazení diagramu. Lišty se také automaticky zobrazí, pokud se naopak zvětší diagram, například kvůli vygenerování většího diagramu. Při zvětšení okna tak, že se do



Obrázek 32. Okno diagramu programu LatVis

něj vleze celý diagram, lišty zmizí. Kliknutím pravého tlačítka na ploše diagramu se zobrazí *popup menu*, hlavní menu pro práci s množinou a diagramem. Popup menu lze zrušit výběrem nějaké jeho položky nebo kliknutím libovolného tlačítka na myši mimo toto menu. Uzavření okna zruší diagram i uspořádanou množinu v něm. Pokud jsou tyto neuložené (změněné), program ze uživatele zeptá, zda chce skutečně okno uzavřít a ztratit tak změny provedené v diagramu a množině.

Diagram uspořádané množiny je zobrazen přes celý obsah okna a je automaticky centrován do jeho středu. Vrcholy diagramu jsou zobrazeny jako černé puntíky se jménem prvku množiny, kterému vrchol patří, umístěném vlevo nahoře od vrcholu v rámečku. Hrany diagramu jsou zobrazeny jako čáry propojující vrcholy. Vrcholy je možné pohybovat tak, že na vrcholu stiskneme levé tlačítko myši, držíme jej stisknuté a pohybujeme myší, nakonec tlačítko myši uvolníme. Je to obdoba klasické metody *drag & drop*. Vrchol není možné posunout mimo okno diagramu. Zároveň je jeho možný posun omezen do pásu mezi nejníže umístěným vrcholem z následovníků jeho prvku a nejvýše umístěným vrcholem z předchůdců jeho prvku, tzn. vrchol prvku není možné posunout nad vrchol libovolného jeho následovníka nebo pod vrchol libovolného jeho předchůdce. To vyplývá z potřeby dodržet definici Hasseova diagramu uspořádané množiny. Pokud máme v diagramu vybrané nějaké vrcholy (viz dále), můžeme pohybovat všemi zároveň stejným postupem jako u jednoho vrcholu (libovolného vybraného), ale navíc držíme ještě stisknutou klávesu $\boxed{\text{SHIFT}}$.

Popup menu okna diagramu:

File

stejné menu jako z hlavního menu programu, ale navíc má položky pro uložení množiny (a diagramu) a zavření okna diagramu

Save CTRL+S

uloží množinu a diagram do souboru, ze kterého byla načtena nebo předtím uložena, pokud jde o novou množinu, která ještě nebyla uložena, provede se stejná akce, jakou prování následující položka *File→Save As...*

Save As...

uložení množiny a diagramu, vyvolá standardní systémový dialog uložení souboru, nazvaný *Save set*.

Close CTRL+W

zavře okno diagramu, má stejnou funkci jako standardní zavírací tlačítko okna a provádějí se proto i stejné činnosti jako při zavření tohoto okna

Edit

obsahuje standardní operace editace jako vrácení (undo), zrušení vrácení (redo), kopírování, vyříznutí a vložení vrcholů a smazání diagramu

Undo CTRL+Z

provádí standardní operaci vrácení (undo), vrací se poslední změna množiny nebo diagramu od nynějšího stavu

Redo CTRL+R

provádí standardní operaci zrušení vrácení (redo), zruší se vrácení poslední změny množiny nebo diagramu, tj. obnoví se původní stav před vrácením

Cut CTRL+X

provádí standardní operaci vyříznutí vrcholů z diagramu i odpovídajících prvků uspořádané množiny (včetně jejich vztahů), vrcholy a prvky jsou z diagramu a množiny smazány a zkopírovány do pomyslné schránky, předchozí obsah schránky je ztracen, operace pracuje s vybranými vrcholy, viz položka *Select elements*

Copy CTRL+C

provádí standardní operaci kopírování vrcholů diagramu i odpovídajících prvků uspořádané množiny (včetně jejich vztahů), vrcholy a prvky jsou zkopírovány do pomyslné schránky, operace pracuje s vybranými vrcholy

Paste $\boxed{\text{CTRL+V}}$
: provádí standardní operaci vložení vrcholů z pomyslné schránky do diagramu a množiny, schránka je společná pro všechna okna diagramu, lze proto kopírovat vrcholy z jednoho diagramu do jiného, nelze vložit vrchol prvku se jménem, které již má nějaký prvek v cílové množině

Clear $\boxed{\text{CTRL+K}}$
: smaže diagram i všechny prvky množiny, do pomyslné schránky se nic nekopíruje

Select

obsahuje standardní operace práce s výběrem, invertovat výběr, vybrat vše a zrušit výběr, navíc obsahuje položky výběru označených vrcholů (viz dále) a položku samotného výběru vrcholu

Invert $\boxed{\text{CTRL+I}}$
: invertuje výběr vrcholů, tj. vybere doplňkovou podmnožinu vrcholů k podmnožině vybraných vrcholů z celkového počtu vrcholů diagramu

All $\boxed{\text{CTRL+A}}$
: vybere všechny vrcholy diagramu

None $\boxed{\text{SHIFT+CTRL+A}}$
: zruší výběr vrcholů, tj. po provedení operace nebude vybrán žádný vrchol

Marked $\boxed{\text{CTRL+M}}$
: vybere označené vrcholy

Select elements $\boxed{\text{S}}$
: zapne (implicitní) mód programu výběru vrcholů, této položce odpovídá tlačítko *Select* v hlavním okně programu, kliknutí na vrchol diagramu se stisknutou klávesou $\boxed{\text{SHIFT}}$ tento vrchol vybere a ten se zbarví zeleně

View

obsahuje položku zobrazení diagramu v novém okně

New Diagram View
: zobrazí diagram množiny v novém okně diagramu, diagram ani množina se nekopírují, okno pracuje se stejným diagramem i množinou jako původní okno, jsou tedy navzájem zcela zaměnitelné a rovnocenné, akce provedená v jednom okně se projeví i ve druhém, libovolné původní nebo nové okno je možné zavřít, to druhé zůstává

Data

obsahuje položky vytvoření nového prvku množiny a vrcholu diagramu, přejmenování prvku, smazání prvku a vrcholu, definování uspořádání dvou prvků a zrušení tohoto uspořádání a podmenu označování prvků

New elements $\boxed{\text{INSERT}}$

zapne mód programu vkládání prvků do množiny a vrcholů do diagramu, této položce odpovídá tlačítko *New* v hlavním okně programu, kliknutí na prázdné místo v okně diagramu vyvolá dialog *New element* pro vložení nového prvku, vrchol nového prvku je umístěn přibližně do místa kliknutí nebo je alespoň zachována relativní pozice místa kliknutí vzhledem k ostatním vrcholům, nesmíme totiž zapomínat na automatické centrování diagramu v okně

Rename element...

přejmenování prvku, na jehož vrcholu bylo popup menu vyvoláno, pro správnou funkci tedy musíme nejdříve menu vyvolat kliknutím pravým tlačítkem myši na vrcholu prvku, který chceme přejmenovat, vyvolá dialog *Rename element*

Remove elements $\boxed{\text{DELETE}}$

zapne mód programu odebírání prvků z množiny a vrcholů z diagramu, této položce odpovídá tlačítko *Remove* v hlavním okně programu, kliknutí na vrchol diagramu smaže vrchol z diagramu a prvek z množiny, s prvkem a vrcholem se samozřejmě smažou i hrany spojující tento vrchol s ostatními a vztahy s tímto prvkem v množině

Define cover $x < y$ $\boxed{\text{C}}$

zapne mód programu definování uspořádání dvou prvků množiny, této položce odpovídá tlačítko *Define cover* v hlavním okně programu, po zapnutí módu se klikne nejdříve na vrchol prvku x, pak na vrchol prvku y a následně se vytvoří v množině vztah $x < y$ mezi těmito prvky a v diagramu se zobrazí hrana propojující vrcholy, předchozí platí, pouze pokud je vrchol prvku x v diagramu níže (nebo na stejné úrovni) než vrchol prvku y, v opačném případě se vytvoří vztah $y < x$, tak aby byla dodržena definice Hasseova diagramu, vztah ani hrana se nevytvoří, pokud již v množině existuje mezi prvky tento vztah nebo vztah opačný nebo tranzitivní, při vytvoření hrany se z diagramu automaticky smažou hrany případně vzniknuvších tranzitivních vztahů

Remove cover $\boxed{\text{SHIFT+C}}$

zapne mód programu rušení uspořádání dvou prvků množiny, této položce odpovídá tlačítko *Remove cover* v hlavním okně programu, po zapnutí módu se klikne nejdříve na první vrchol, pak na druhý vrchol a následně se v množině zruší vztah mezi prvky těchto vrcholů a z diagramu se vymaže hrana propojující tyto vrcholy, pokud jsou vrcholy na stejné úrovni, musí se kliknout nejdříve na vrchol menšího prvku, aby byl vztah zrušen, vztah ani hrana se nezruší, pokud je tento vztah mezi prvky tranzitivní, při zrušení vztahu se v diagramu automaticky vytvoří hrany vztahů, které byly před zrušením vztahu tranzitivní a nyní po zrušení vztahu již nejsou

Mark

toto podmenu obsahuje položky označování vrcholů různých podmnožin prvků, označené vrcholy jsou zbarveny modře, označené a zároveň vybrané vrcholy jsou zbarveny modrozeleně (kyanově)

Lower elements SHIFT+L

zapne mód programu označení vrcholů předchůdců prvku, této položce odpovídá tlačítko *Mark lower* v hlavním okně programu, kliknutí na vrchol prvku v diagramu označí vrcholy předchůdců tohoto prvku

Upper elements SHIFT+U

analogicky předchozí položce pro následovníky, této položce odpovídá tlačítko *Mark upper* v hlavním okně programu

Minimal elements SHIFT+I

označí vrcholy minimálních prvků z podmnožiny prvků vybraných vrcholů

Maximal elements SHIFT+A

analogicky předchozí položce pro maximální prvky

Least element SHIFT+E

analogicky pro nejmenší prvek

Greatest element SHIFT+G

analogicky pro největší prvek

Lower cone SHIFT+W

analogicky pro dolní kužel

Upper cone SHIFT+R

analogicky pro horní kužel

Infimum SHIFT+F

analogicky pro infimum

Supremum SHIFT+S

analogicky pro supremum

Paths... SHIFT+P

zapne mód programu označení vrcholů prvků na cestě mezi dvěma prvky množiny, této položce odpovídá tlačítko *Mark paths* v hlavním okně programu, po zapnutí módu se klikne nejdříve na první vrchol prvku, pak na druhý vrchol prvku a následně se v diagramu označí vrcholy těchto prvků a vrcholy prvků ležících na cestě mezi těmito prvky, pokud mezi prvky existuje více různých cest, vyvolá se dialog se seznamem všech cest nazvaný *Select path*, ve kterém je možné vybrat cestu, jejíž vrcholy se mají označit, pokud naopak mezi prvky neexistuje žádná cesta, neoznačí se žádné vrcholy

Unmark all $\boxed{\text{SHIFT+M}}$

zruší označení vrcholů, tj. po provedení operace nebude označen žádný vrchol

Diagram

obsahuje položku generování diagramu

Rearrange diagram... $\boxed{\text{CTRL+D}}$

stěžejní položka programu, generování diagramu uspořádané množiny, vyvolá dialog *Rearrange diagram*

Dialogy spouštěné z položek menu:

Save set

Standardní systémový dialog uložení souboru, vyvolán z menu *File*→*Save As...*, nemodální. Dialog se v mnohém podobá standardnímu systémovému dialogu otevření souboru, přesněji řečeno oba dialogy jsou prakticky stejné.

Typy souboru v gtkmm verzi jsou: podle přípony souboru (*By Extension*), nativní (*LAT (LatVis)*), zdrojový soubor METAPOSTu (*MetaPost*), standardní přípona *.mp*, a Zapouzdřený PostScript (*Encapsulated PostScript*) se standardní příponou souboru *.eps*. U posledních dvou se vlastně jedná o export diagramu, množina není ukládána.

V Qt verzi jsou možnosti souborového filtru: všechny soubory (*All files (*)*), nativní typ souboru (*LAT (LatVis) files (*.lat)*), zdrojové soubory METAPOSTu (*MetaPost files (*.mp)*) a Zapouzdřený PostScript (*Encapsulated PostScript files (*.eps)*).

Dialog lze opět potvrdit až při zadání nebo zvolení jména souboru, také i klávesou $\boxed{\text{ENTER}}$, pokud je fokus na editačním políčku jména souboru. Po potvrzení dialogu je množina i s diagramem uložena do souboru. Pokud se během ukládání souboru vyskytne nějaká chyba, je zobrazen (modální) dialog s hláškou vysvětlující nastalou chybu.

New element

Dialog vložení nového prvku do množiny a vrcholu do diagramu, vyvolán kliknutím na prázdné místo v okně diagramu při zapnutém módu vkládání prvků a vrcholů, modální.

V dialogu se zadává pouze jméno nového prvku v editačním políčku *New element's name*, které má fokus hned při zobrazení dialogu, aby bylo možné rovnou psát jméno prvku. Dialog nelze potvrdit, dokud není zadáno nějaké jméno (nějaký text) a zároveň pokud zadané jméno má již některý existující prvek v množině. Musíme zadat jméno různé od jmen všech prvků množiny, nelze tedy vytvořit více prvků se stejným jménem.

Potvrdit jej lze i klávesou ENTER, pokud je fokus na editačním políčku. Po potvrzení dialogu je vrchol prvku vložen do diagramu jak bylo popsáno výše.

Rename element

Dialog je totožný s dialogem *New element*, je vyvolán z položky popup menu *Data→Rename element...*

V dialogu se zadává nové jméno prvku, po potvrzení dialogu je jméno prvku změněno na toto nové jméno.

Select path

Dialog výběru cesty mezi dvěma prvky množiny a označení vrcholů prvků na cestě v diagramu, vyvolán kliknutím na dva vrcholy prvků, mezi nimiž existuje v množině více různých cest při zapnutém módu označení vrcholů prvků na cestě mezi dvěma prvky, dialog je nemodální.

V dialogu je zobrazen seznam *Found paths* cest mezi dvěma krajními prvky každé položky seznamu. Cesty jsou vypsány v podobě řetězce jmen prvků na cestě oddělených pomlčkou. Při vybrání položky seznamu, tj. cesty, se v diagramu označí vrcholy prvků cesty.

Rearrange diagram

Dialog generování (nebo úpravy) diagramu uspořádané množiny, vyvolán z položky popup menu *Diagram→Rearrange diagram...*, dialog je nemodální.

Dialog obsahuje tři stránky zobrazené pomocí záložek. Každá záložka je postupně pro metodu generování diagramu úrovňovou (*Level*), geometrickou (*Geometric*) a vrstvovou (*Layer method*). Na straně každé záložky jsou grafické prvky pro nastavení parametrů odpovídající metody generovaní diagramu.

Na stránce úrovňové metody je přepínací tlačítko pro nastavení, zda se má diagram generovat od minimálních prvků nebo opačně (*Generate diagram starting with minimal elements*).

Na stránce geometrické metody (předvybrané) je přepínací tlačítko pro nastavení, zda se má diagram generovat od největšího prvku nebo opačně (*Generate diagram starting with the greatest element*).

Na stránce vrstvové metody je již parametrů více. Přepínací tlačítko pro nastavení, zda se má diagram generovat od maximálních prvků nebo opačně (*Generate diagram starting with maximal elements*). Tlačítka pro výběr metody rozvržení do vrstev (*Layering*), metody nejdelší cesty (*Longest Path*) nebo Coffman-Grahamovy metody (*Coffman-Graham*). Editační políčko pro zadání kladného čísla udávajícího maximální šířku při rozvržení do vrstev Coffman-Grahamovou metodou (*Maximal width of the Coffman-Graham layering*). Při zadání nekladného čísla se při potvrzení dialogu zobrazí dialog s varovnou hláškou. Dále jsou zde tlačítka pro výběr datové reprezentace řetězců falešných vrcholů v diagramu (*Dummy chains internal data representation*), možnosti výběru jsou stejné, jako u dialogu *New set*. Nakonec jsou zde ještě tlačítka pro výběr metody redukce křížení (*Crossing reduction*), možnosti jsou třídění prohozením sousedů (*Adjacent-Exchange sorting*), třídění rozdělením (*Split sorting*), metoda průměru (*Barycenter*) a metoda mediánu (*Median*).

Po potvrzení dialogu (tlačítkem *Rearrange*) je diagram uspořádané množiny vygenerován (upraven) a zobrazen v okně diagramu. Pro generování je použita metoda aktuálně vybrané stránky záložek a její nastavení. Dialog se po potvrzení nezavírá, je umožněno opětovné generování diagramu pro různé metody a jejich různá nastavení pro porovnání výsledku generování, bez opakovaného vybírání položky menu *Diagram→Rearrange diagram...*.

4.5. Exportní formáty

Po vygenerování diagramu a jeho případném upravení (posunem vrcholů) do výsledné podoby je možné jej uložit spolu s uspořádanou množinou do souboru. Kromě tohoto uložení je možné samotný diagram ještě exportovat do dvou vektorových formátů popisu obrázků. Obrovskou výhodou těchto formátů jsou jejich široké možnosti dalšího zpracování bez ztráty kvality obrázku, např. zvětšování do prakticky libovolné velikosti.

4.5.1. Nativní XML

V první řadě lze uspořádanou množinu spolu s diagramem uložit do nativního formátu programu. Tímto formátem je značkovací metajazyk *XML*. Informace z programu se tedy ukládají jako *XML dokument*. Tento dokument musí mít definovánu svoji strukturu, tím se z metajazyka XML stane jazyk dokumentu. Strukturu XML dokumentu můžeme definovat pomocí definice typu dokumentu (*Document Data Type Definition - DTD*). Následuje DTD pro dokument programu *LatVis*.

```
<!ELEMENT ordered_set (elements?, orders?, diagram?)>
<!ATTLIST ordered_set data_type (array|list|tree) #REQUIRED>

<!ELEMENT elements (element*)>

<!ELEMENT element (vertex?)>
<!ATTLIST element name CDATA #REQUIRED>

<!ELEMENT vertex (x, y)>
<!ATTLIST vertex dummy_id NMTOKEN #IMPLIED>

<!ELEMENT x (#PCDATA)>

<!ELEMENT y (#PCDATA)>

<!ELEMENT orders (order*)>

<!ELEMENT order (element, element)>

<!ELEMENT diagram (dummies?, edges?)>

<!ELEMENT dummies (dummy*)>

<!ELEMENT dummy (x, y)?>
<!-- x, y are required if dummy is under dummies -->
<!ATTLIST dummy id NMTOKEN #REQUIRED>

<!ELEMENT edges (edge*)>

<!ELEMENT edge (dummy, dummy)>
```

Program ukládá a načítá soubory tohoto formátu s implicitní příponou souboru *.lat*. Pro načítání a ukládání XML souborů je použita knihovna *libxml2* (`http://xmlsoft.org`).

4.5.2. MetaPost

Diagram uspořádané množiny vyrobený pomocí tohoto programu je možné exportovat do formátu zdrojového souboru jazyka METAPOST. METAPOST (`http://cm.bell-labs.com/who/hobby/metapost.html`) je programovací jazyk pro popis a následné kreslení obrázků. Jeho autorem je John D. Hobby. Po zpracování zdrojového souboru překladačem tohoto jazyka je výsledkem program v jazyce *PostScript*, který vyrenderuje daný obrázek. Na rozdíl od PostScriptu poskytuje ale mnoho zajímavých funkcí, díky kterým je možno kreslení zjednodušit a zrychlit. Vygenerované obrázky je možno použít mnoha způsoby, mimo jiné je možno propojit je s TEXovským textem. Kvalita obrázků popsaných v jazyku METAPOST je velmi vysoká.

Jako příklady obrázků v tomto jazyku a ukázky jeho možností mohou sloužit veškeré obrázky diagramů a grafů v této dokumentaci, které jsou všechny vytvořeny právě v METAPOSTu.

Program exportuje soubory tohoto formátu se standardní implicitní příponou souboru *.mp*.

4.5.3. Zapouzdřený PostScript

Protože překladač jazyka METAPOST je součástí velkého balíku typografického systému TEX a tento balík většinou nebývá na systémech Windows standardně nainstalován, rozhodl jsem se implementovat i jednoduchý export přímo do *PostScriptu*. Výsledek exportu rozhodně není takové kvality, jaké je možné dosáhnout překladem obrázku zapsaného v METAPOSTu, ale ve většině případů je možné jej s úspěchem použít pro prezentaci diagramů uspořádané množiny a svazu. Konkrétně je generován program jazyka *Zapouzdřeného PostScriptu* (*Encapsulated PostScript, EPS*) verze 3.0.

Program exportuje soubory tohoto formátu se standardní implicitní příponou souboru *.eps*.

5. Závěr

V této práci jsem se snažil prozkoumat, implementovat a porovnat různé metody generování Hasseova diagramu uspořádaných množin a svazů. Těchto metod ale existuje nepřeberné množství, takže metody implementované ve výsledném programu představují jen nepatrný zlomek z tohoto množství. Proto jsem nastínil i další metody, které generují kvalitní diagramy množin. Tyto další metody a ještě spousta dalších vytvářejí prostor pro rozšíření programu se snahou produkovat stále lepší a lepší diagramy.

Kromě detailního popisu samotných metod a jejich porovnání jsem také rozebral možné datové reprezentace uspořádaných množin a svazů, které jsou v programu také implementovány. Z rozboru těchto reprezentací vyplývá, že další možné rozšíření programu v tomto směru znamená kombinaci obou přístupů reprezentace, rychlého, leč paměťově náročného dvourozměrného pole a pomalého, ale paměťově šetrného seznamu seznamů a binárního vyhledávacího stromu.

Metody použité v programu jsou s úspěchem použitelné pouze do určité velikosti uspořádané množiny nebo svazu. Se vzrůstajícím počtem prvků svazu se ale ukazuje potřeba nového přístupu k problematice: aby diagram odrážel strukturální analýzu svazu, což často znamená dekompozici na menší a snáze zpracovatelné části. Pak by měl být účinný algoritmus založený na dostatečně mnoha grafických vzorech a metodách jejich transformace a kombinace. Důležitý je také aspekt ovlivnění procesu generování uživatelem. Tedy je žádoucí interaktivní implementace.

Resumé

This program implements and compares various methods of the automatic generation of the Hasse diagrams of ordered sets and lattices by means of a computer. The diagram is generated using only the informations defining the ordered set, i.e. the list of its elements and the order relation. No initial diagram is needed. For the resulting diagram, I tried in the implementation of methods to achieve good level of its readability while satisfying common conventions a constraints. The methods used for diagram generation are among well-known layer drawing methods, somehow less known geometrical method, and in addition I designed and implemented my own method. Diagrams generated by this program are of quite good aesthetics criteria, however, you can always adjust the diagram to suit your needs and taste. Besides diagram generation methods, also various data representations of ordered sets are compared.

Reference

[1] Britz, Thomas, Cameron, Peter: *Partially ordered sets*, elektronický článek, 2001

[2] Cole, Richard: *Automated Layout of Concept Lattice Using Layer Diagrams and Additive Diagrams*, elektronický článek, Griffith University, Gold Coast, Australia

[3] Di Battista, Giuseppe, Eades, Peter, Tamassia, Roberto, Tollis, Ioannis G.: *Graph Drawing: Algorithms for the Visualisation of Graphs*, Prentice-Hall, Upper Saddle River, New Jersey, 1999

[4] Ganter, Bernhard, Wille, Rudolf: *Formal Concept Analysis: Mathematical Foundations*, Springer-Verlag, New York, 1997

[5] Horák, Pavel: *Algebra a teoretická aritmetika I.*, skriptum MU, Přírodovědecká fakulta, Brno, 1994

[6] Kaufmann, Michael, Wagner, Dorothea: *Drawing Graphs Methods and Models*, elektronický článek, 2000

[7] Krutský, František: *Algebra I.*, skriptum UP, Přírodovědecká fakulta, Olomouc, 1992

[8] Luksch, Peter, Skorsky, Martin, Wille, Rudolf: *On Drawing Concept Lattices With a Computer*, elektronický článek, Forschungsgruppe Begriffsanalyse Fachbereich Mathematik, Technische Hochschule Darmstadt

[9] Rachůnek, Jiří: *Algebra a teoretická aritmetika* , skriptum UP, Přírodovědecká fakulta, Olomouc, 1983

[10] Skorsky, Martin: *Reihe Mathematik: Endliche Verbände Diagramme und Eigenschaften*, Verlag Shaker, Aachen, Germany, 1992

[11] Stumme, Gerd, Wille, Rudolf: *A Geometrical Heuristic for Drawing Concept Lattices*, elektronický článek, Technische Hochschule Darmstadt, Fachbereich Mathematik, Springer-Verlag, Berlin-Heidelberg, 1995

[12] Vychodil, Vilém: *Úvod do finitní matematiky*, elektronický text, 2001

[13] Wille, Rudolf: *Lattices In Data Analysis: How To Draw Them With a Computer*, elektronický článek, Fachbereich Mathematik, Technische Hochschule Darmstadt

A. Příloha CD-R

Na přiloženém CD-R jsou uloženy kompletní zdrojové texty programu a této dokumentace, vše potřebné pro bezproblémový překlad programu, testovací ukázky vytvořených diagramů, text této dokumentace a přeložený program, spustitelný přímo z CD-R.

Následuje adresářová struktura CD-R a stručný popis obsahu každého adresáře a souboru:

binaries/
: Binární soubory přeloženého programu, spustitelné přímo z CD-R, a všechny potřebné knihovny a lokalizační soubory jazykových mutací programu (v podadresáři **locale/**). Soubory jsou v podadresářích podle operačního systému.

  linux.i386.static/
  : Program přeložený pro operační systémy s jádrem Linux na platformě i386. Soubor `latvis-gtkmm` je program s gtkmm verzí GUI, soubor `latvis-qt` je program s Qt verzí GUI.

  win32.i386/
  : Program přeložený pro 32-bitové operační systémy Windows na platformě i386, soubor programu je `latvis.exe`.

docs/
: Tato dokumentace diplomové práce ve formátech PostScript (soubor `latvis.ps`) a PDF (soubor `latvis.pdf`).

  articles/
  : Elektronické články a texty, na které je v této dokumentaci odkazováno.

examples/
: Testovací příklady uspořádaných množin a jejich diagramů. Soubory jsou v podadresářích **ISO-8859-2/** a **CP1250/**, v kódování odpovídajícím jménu adresáře.

requirements/
: Zdrojové nebo předkompilované původní balíky softwaru, který je potřeba mít nainstalovaný v operačním systému pro překlad programu. Soubory jsou v podadresářích podle operačního systému.

sources/

Balíky kompletních zdrojových textů programu. Soubor `latvis-x.y.z.tar.gz` je pro unixové systémy, soubor `latvis-x.y.z-win.zip` pro systémy Windows. *x.y.z* označuje verzi programu. Po rozbalení balíku si přečtěte soubor `README` nebo `README.windows`. Soubor `windows_requirements.zip` je balík hlavičkových souborů, knihoven a programů potřebných pro překlad programu pod systémy Windows.

`README`, `README.windows`

Základní instrukce pro instalaci a spuštění programu. První je pro unixové systémy, druhý pro systémy Windows.

`COPYING.ISO-8859-2`, `COPYING.CP1250`

Licence programu v odpovídajícím kódování.