Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Vizualizácia vlastností grafov bez násobných hrán a slučiek

Bakalárska práca

2016 Peter Becza

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Vizualizácia vlastností grafov bez násobných hrán a slučiek

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	doc. RNDr. Andrej Ferko, PhD

Bratislava, 2016 Peter Becza

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Peter Becza
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Vizualizácia vlastností grafov bez násobných hrán a slučiek
Graph Properties Visualization

Cieľ: Cieľom práce je implementovať vizualizované vybrané poznatky z teórie grafov v rovine, navrhnúť vizualizačné postupy pre neorientované grafy s n-vrcholmi (bez multihrán a slučiek) a ofarbenie častí roviny podľa výskytu vybraných vlastností.

Literatúra:
DIESTEL, Reinhard Teória grafov, Springer, slovensky preklad Peter Hrasko, FMFI UK Bratislava
Pavol KORINEK: Vizualizácia vybraných poznatkov teórie grafov, diplomova praca, FMFI UK, Bratislava 2007.
HOLOTŇÁK, Ondrej, 2000, Zbierka úloh z teórie grafov, diplomova praca, FMFI UK, Bratislava 2000.
ŠKOVIERA, Martin, Materiálny k predmetu Úvod do kombinatoriky a teórie grafov
Benjamin B. Bederson and Ben Shneiderman (2003). The Craft of Information Visualization: Readings and Reflections, Morgan Kaufmann ISBN 1-55860-915-6.

Anotácia:
1. Prehľad problematiky.
2. Špecifikácia projektu.
3. Implementácia.

Kľúčové slová: teória grafov, vizualizácia

Vedúci: doc. RNDr. Andrej Ferko, PhD.
Katedra: FMFI.KAGDM - Katedra algebry, geometrie a didaktiky matematiky
Vedúci katedry: prof. RNDr. Pavol Zlatoš, PhD.

Dátum zadania: 26.10.2015

Dátum schválenia: 29.10.2015 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Chcel by som sa poďakovať svojmu školiteľovi Andrejovi Ferkovi za jeho trpezlivosť.

Abstrakt

Teória grafov ako súčasť diskrétnej matematiky je rýchlo rozvíjajúcou sa oblasťou matematiky. Teória grafov skúma abstraktné matematické štruktúry zvané grafy, ktorých pomocou sa rieši mnoho problémov. S nárastom informácií a vedomostí o grafoch je potrebné tieto dáta efektívne spracovávať. Jedným zo spôsobov spracovania dát je vizualizácia. V tejto bakalárskej práci budeme definovať pozíciu grafu v 2D priestore a vytvoríme nástroj, ktorý dokáže postupne lexikograficky generovať konečné neorientované grafy bez multihrán a slučiek. Dokáže ich zobraziť v rovine a priradí im farbu podľa toho, či graf má vybranú vlastnosť alebo nie. Prostriedkom takejto vizualizácie bude jednoduchšie uvedomiť pozíciu konkrétneho grafu vo vzťahu k ostatným grafom, čo môže pomôcť pri budúcom skúmaní ich vlastností.

Kľúčové slová: graf, generovanie grafov, vizualizácia, mapovanie, farbenie grafov, spracovanie dát

Abstract

Graph theory as part of discrete mathematics is a rapidly developing area of mathematics. Graph theory explores abstract mathematical structures called graphs. Many different problems can be modeled using graphs.It is necessary to process data efficiently. One of many ways of data processing is visualization. In this bachelor thesis we define graph position in 2D space. We will create a tool that can progressively generate final undirected graphs in lexicographicall order without multiedges and loops. Tool will display this graphs in 2D plane and assign them color depending on whether the graph has our choosen property or not. This visualization means it will be easier to recognize and understand the position of a particular graph in relation to other graphs, which may assist in future analysing and hopefully estimating of their properties.

Keywords: graph, graph generating, visualization, mapping, graph coloring, data processing

Obsah

Zoznam obrázkov

Úvod

Teória grafov ako súčasť diskrétnej matematiky je modernou a rýchlo sa vyvíjajúcou oblasťou matematiky, ktorá je úzko spätá s teoretickou a aplikovanou informatikou.

Môžeme rozlišovať veľké množstvo druhov grafov. V tejto práci nebudeme hovoriť o grafoch funkcií, či štatistických grafoch, ale o grafoch ako o abstraktných matematických štruktúrach. Pre jednoduchosť si ich možno predstaviť ako body (vrcholy) pospájané čiarami (hrany). Formálne grafy zadefinujeme v kapitole 1.

Budeme hovoriť o konečných neorientovaných grafoch, teda takých, ktoré majú konečný počet vrcholov a hrany vedúce z jedného vrchola do druhého sú obojsmerné. Nebudeme uvažovať o grafoch, ktoré obsahujú slučky, teda hrany z vrchola do seba samého a takisto nebudeme uvažovať o tom, že by medzi dvomi vrcholmi bolo viac hrán. Teda medzi dvomi vrcholmi buď hrana nie je alebo je práve jedna.

Grafy pomáhajú napr. riešiť problémy s najkratšou cestou medzi dvomi mestami alebo sa na ne dajú previesť vzťahy medzi entitami, či pomocou nich dokážeme simulovať chemické vzorce. Mnoho problémov z reálneho sveta je možné previesť na problémy grafu, preto význam grafov v matematike v posledných desaťročiach narastá [1]. Hoci sa grafové úlohy a problémy môžu zdať jednoduché, každý aj čiastkový výsledok má veľký význam.

História a významné osobnosti

Korene diskrétnej matematiky a teórie grafov siahajú do 18. a 19. storočia.

Za jedného zo zakladateľov teórie grafov je považovaný Leonard Euler, ktorý v roku 1736 publikoval svoje riešenie problému Siedmich mostov mesta Königsberg [2]. Skúmal, či je možné prejsť každým zo siedmich mostov práve raz tak, aby sa vrátil na východziu pozíciu.

V roku 1847 sa Gustav Kirchhoff venujúci sa topológii elektrických sietí skúmal počet kostier grafu [3]. Niekoľko rokov neskôr, v roku 1857, Arthur Cayley pomocou grafov opisoval chemické zlúčeniny. Skúmal alkány, uhľovodíky, ktoré medzi svojimi molekulami neobsahujú žiadne násobné väzby [4].

Ďalšou významnejšou osobnosťou bol William Hamilton, ktorý v roku 1859 začal skúmať cesty a kružnice v grafoch. Vytvoril hru The Icosian Game, v ktorej bolo úlohou zisťovať, či graf obsahuje hamiltonovskú kružnicu [5].

Jednou z najznámejších úloh 19. storočia bol problém štyroch farieb. Teda, či nám stačia len 4 farby na ofarbenie ľubovolnej mapy. Tento problém bol úspešne vyriešený a dokázaný v roku 1976 [6].

Osobnosti teórie grafov sú zastúpené aj v našich končinách. Jednou z najznámejších osobnosti Československa v tejto oblasti bol Otakar Borůvka, ktorý v roku 1926 publikoval svoj algoritmus pre nájdenie minimálnej kostry grafu [7]. Tento algoritmus rieši problém čo najvýhodnejšej výstavby elektrických sietí.

Motivácia

Keď sa uvažuje o grafoch, hovorí sa o konkrétnom grafe, o grafových vlastnostiach, o druhoch grafov, a pod., lenže s nárastom informácií klesá prehľadnosť dát, preto vznikla myšlienka pokúsiť sa zobraziť grafy do priestoru, a zadefinovať tak akýsi grafový priestor. Dosiahneme tým to, že graf ako taký, už nebude len ako abstraktná entita niekde v neznáme, ale už si ho budeme môcť predstaviť v priestore.

S touto myšlienkou prišiel študent Ernest Štibrányi vo svojej diplomovej práci [8], ktorý sa pokúsil zobraziť neorientované grafy v 2D priestore. Takéto zobrazenie prináša nový pohľad na grafy. Tým, že takéto grafy budú ofarbené podľa vlastností, ktoré boli na nich analyzované, dokážeme zadefinovať pozíciu konkrétneho grafu v priestore.

Z akademického hľadiska prináša takýto grafový priestor študentom lepší prehľad o tom, kde sa graf, ktorý skúmajú, v danej vizualizácii nachádza. Vďaka vizualizácii je ľahšie pochopenie problematiky konkrétnych vlastností grafov. Takáto definícia grafového priestoru dokáže ukázať či odhaliť v priestore výskyt grafov s nejakou vlastnosťou, čím môže výskumníkovi efektívnejšie uľahčiť prácu. Vizualizácia môže dopomôcť skúmateľovi v tom, aby si všimol vzorky (pattern) výskytu danej vlastnosti grafu.

Cieľ práce

Cieľom práce je implementovať vizualizované vybrané poznatky z teorie grafov v 2D priestore, navrhnúť vizualizačné postupy pre neorientované grafy s n-vrcholmi (bez multihrán a slučiek) a ofarbenie časti roviny podľa výskytu vybraných vlastností.

Súvisiace práce

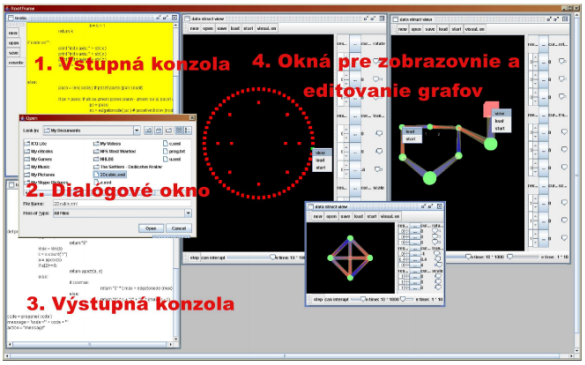
Teraz spomnieme niekoľko podobných prác súvisiacich s tou našou.

Nástroj *uDraw(Graph)* je nástroj na vizualizáciu grafov so širokou funkcionalitou. Dokáže automaticky zobraziť graf, vytvoriť mapovanie či hierarchiu vizualizácie. Tento nástroj je možné použiť vo vlastnej implementácii programu.
www.informatik.uni-bremen.de/uDrawGraph/en/uDrawGraph/uDrawGraph.html

Na vizualizáciu dát a grafov v programovacom jazyku Java dobre slúží nástroj *Drawing Graphs with VGJ* [9]. Zvláda zobrazovanie grafov a grafových schém. Ponúka rôzne spôsoby importu dát.

S myšlienkou zobrazovanie grafov do roviny prišiel Ernest Štibrányi vo svojej diplomovej práci *Vizualizácia teórie grafov* [8], kde dokázal generovať všetky neorientované grafy (bez hrán a slučiek) až do počtu vrcholov 4. Ku grafom s väčším počtom vrcholov sa nedostal. Program k práci je písaný v programovacom jazyku C.

V práci *Vizualizácia vybraných algoritmov a vlastnosti z teórie grafov* [10] autor Pavol Korínek vytvoril nástroj *Grapher* (Obrázok 1), ktorý dokázal generovať grafy do 3D priestoru, a znázorňovať na nich vybrané vlastnosti.



Obr. 1: Ukážka architektúry programu Grapher [10]

Hlavným programovacím jazykom bol jazyk Java, v ktorom bolo vytvorené použí-

vateľské prostredie, práca s grafmi a vizualizácia, kým výpočtová časť vola napísaná v jazyku Python.

Metodika práce

V tejto práci vytvoríme nástroj, ktorým budeme zobrazovať grafy ako body v rovine, tj. v Euklidovskom priestore. Tieto body budú ofarbené podľa toho, ktorú z nami vybraných skúmaných vlastností majú. Výsledný obrázok sa generuje podľa vopred definovaných pravidiel.

Budeme generovať všetky grafy z počiatočnej konfigurácie do nasledujúcej v lexikografickom poradí. Prácu rozdelíme na 3 hlavné fázy:

1. generovanie grafu
2. analýza grafu
3. zobrazenie grafu

Bude potrebné skombinovať časť práce mojich predchodcov s kladením dôrazu na jednoduchosť a prehľadnosť. Nástroj má slúžiť ako základný nástroj, ktorý dokáže efektívne a jednoducho zobraziť výsledok, ale s možnosťou ďalšieho rozširovania a vylepšovania do budúcnosti. Tento nástroj bude vhodný pre ďalšiu úpravu, pre pridávanie nových vlastností, rozširovanie funkcionality a jej zefektívňovanie.

Kapitola 1

Úvod do teórie grafov

Krátky úvod do problematiky teórie grafov. V tejto kapitole si zadefinujeme základné pojmy pre prácu s grafmi. V práci uvažujeme len o neorientovaných grafoch, preto budeme definovať všetko potrebné len pre tento typ grafov.

1.1 Základné definície a tvrdenia

Definície, vety a tvrdenia aj keď popisujú rovnakú skutočnosť, líšia sa formuláciou v závislosti od prostredia a súvislostí, kde sa používajú. Kým v literatúre [11] venujúcej sa grafom z pohľadu sietí a ich aplikácii hovoríme o uzloch, v teórii grafov podľa [12] a [13] používame pojem vrchol. Budeme sa držať druhej verzie, nakoľko sa naša práca viac približuje k tejto problematike.

1.1.1 Grafy

Definícia 1.1.1.1. Graf G je dvojica $G = (V, E)$ disjunktných množín, kde prvky množiny E sú 2-prvkové podmnožiny množiny V. Platí, že $E \subseteq |V^2|$.

Definícia 1.1.1.2. Graf $G' = (V', E')$ je podgraf grafu $G = (V, E)$, ak platí $V' \subset V$ a súčasne $E' \subset (E \cap \binom{V}{2})$.

Definícia 1.1.1.3. Množinu vrcholov grafu G označíme $V(G)$, skrátene V.

Definícia 1.1.1.4. Hrana grafu je neusporiadaná dvojica vrcholov grafu (u, v) taká, že $u, v \in V$. Množinu hrán grafu G označíme $E(G)$, skrátene E.

Definícia 1.1.1.5. Hrana grafu (u, v) je *incidentná* s vrcholom x, ak platí $x \in \{u, v\}$.

Definícia 1.1.1.6. Neorientovaná hrana grafu je neusporiadaná dvojica vrcholov grafu taká, že hranou sa dá prechádzať oboma smermi.

Definícia 1.1.1.7. Komplement grafu $G = (V, E)$ je graf $G' = (V', E')$ pre ktorý platí, že $V = V'$ a pre každú hranu platí, $e \in E'$ práve vtedy, ak $e \notin E$.

1.1.2 Stupeň vrchola

Definícia 1.1.2.1. Nech $G = (V, E)$ je neprázdny graf. Množina susedov vrchola u sa označuje $N_G(u)$ alebo aj $N(u)$.

Definícia 1.1.2.2. Stupeň $d(v)$ vrcholu v je číslo počtu hrán incidentných s vrcholom v. Hodnota stupňa vrchola je rovná počtu susedov daného vrchola. Vrchol stupňa 0 nazývame *izolovaný*.

Definícia 1.1.2.3. Číslo $\delta(G) = min\{d(v)|v \in V\}$ je minimálny stupeň grafu G.
Číslo $\Delta(G) = max\{d(v)|v \in V\}$ je maximálny stupeň grafu G.

Definícia 1.1.2.4. Priemerný stupeň grafu G je číslo

$$d(G) = \frac{1}{|V|} \sum_{v \in V} d(v). \tag{1.1}$$

Platí, že

$$\delta(G) \leq d(G) \leq \Delta(G). \tag{1.2}$$

Definícia 1.1.2.5. Ak v grafe G majú všetky vrcholy rovnaký stupeň k, potom G je *k-regulárny*, skrátene *regulárny*.

Veta 1.1.2.6. Graf má párny počet vrcholov nepárneho stupňa.

1.1.3 Sledy, cesty a cykly

Definícia 1.1.3.1. *Sled* dĺžky k v grafe G je neprázdna striedavá postupnosť $v_0e_0v_1e_1 \dots e_{k-1}v_k$ vrcholov a hrán v G taká, že $e_i = \{v_i, v_{i+1}\}$ pre všetky $i < k$.
Ak $v_k = v_0$, hovoríme, že sled je *uzavretý*.
Ak sú hrany v slede od seba navzájom rôzne, *sled* nazývame *ťah*.

Definícia 1.1.3.2. *Cesta* P v grafe G je neprázdna striedavá postupnosť $v_0e_0v_1e_1 \dots e_{k-1}v_k$ vrcholov a hrán v G taká, že $e_i = \{v_i, v_{i+1}\}$ pre všetky $i < k$ a platí, že všetky vrcholy aj hrany sú navzájom rôzne.

Definícia 1.1.3.3. Dĺžka l cesty P v grafe G je počet hrán cesty P. Označujeme P^l. $l =< 0, |E| >$

Definícia 1.1.3.4. Ak $P = x_0...x_{k-1}$ je cesta a $l \geq 3$, potom graf $C = P + x_{k-1}x_0$ nazývame *cyklus*, resp. *kružnica*.
Počet hrán, resp. vrcholov kružnice navývame *dĺžka* kružnice.
Cyklus dĺžky l označujeme C^l.

Definícia 1.1.3.5. Graf, ktorý neobsahuje cykly nazývame *acyklický*.

Definícia 1.1.3.6. Vzdialenosť $d(x, y)$ v grafe G dvoch vrcholov x, y je dĺžka najkratšej cesty z vrchola x do vrchola y v grafe G. Ak neexistuje cesta z x do y, $d(x, y) = \infty$.

Definícia 1.1.3.7. Priemer grafu $diam(G)$ je najväčšia vzdialenosť medzi ľubovolnými dvoma vrcholmi grafu G.

Veta 1.1.3.8. V každom grafe G obsahujúcom cyklus platí $g(G) \leq 2diam(G) + 1$.

Definícia 1.1.3.9. Centrálnym vrcholom nazveme vrchol v G, ak jeho najväčšia vzdialenosť od ľubovolného iného vrchola je najmenšia možná.
Táto vzdialenosť sa nazýva polomer grafu G, označujeme ho ako $rad(G)$. $rad(G) = min_{x \in V} max_{y \in V} d(x, y)$.

Veta 1.1.3.10. Pre graf G s počtom vrcholov ≥ 3 platí, $rad(G) \leq diam(G) \leq 2rad(G)$.

Veta 1.1.3.11. Graf G s polomerom najviac k a maximálnym stupňom najviac d nemá viac ako $1 + kd^k$ vrcholov.

1.2 Vybrané vlastnosti grafov

V tejto sekcii definujeme niektoré základné vlastnosti grafov, ktoré budeme vizualizovať v tejto práci.

1.2.1 Súvislosť

Definícia 1.2.1.1. Neprázdny graf G nazývame súvislý, ak ľubovoľné dva jeho vrcholy sú spojené cestou v G. Ak $U \subset V(G)$ a $G(U)$ je súvislý, tak aj samotný U nazývame súvislý v G.

Definícia 1.2.1.2. Maximálny súvislý podgraf grafu G sa nazýva *komponent* grafu G.

1.2.2 Stromy

Definícia 1.2.2.1. Acyklický graf nazývame *les*. Súvislý les sa nazýva *strom*.

Veta 1.2.2.2. Nasledujúce tvrdenia sú ekvivalentné pre graf G:

1. G je strom
2. pre všetky dvojice vrcholov v G platí, že sú spojené jedinečnou cestou v G
3. T je minimálne súvislý
4. T je maximálne acyklický

Vlastnosti Súvislý graf s n vrcholmi je strom práve vtedy, keď má $n - 1$ hrán. Každý súvislý graf obsahuje normálnu *kostru* s ľubovoľným vrcholom označeným ako *koreň*.

1.2.3 Eulerovský ťah

Definícia 1.2.3.1. Uzavretý sled v grafe G sa nazýva *eulerovský ťah*, ak prechádza každou hranou grafu G práve raz. Graf, ktorý pripúšťa eulerovský ťah, nazývame *eulerovský*.

1.2.4 Ďalšie vlastnosti grafov

Definícia 1.2.4.1. Graf G nazývame *kubický* práve vtedy, keď každý vrchol grafu je stupňa práve 3.

1.2.5 Reprezentácia grafu

Reprezentácia grafu v programe závisí od konkrétneho problému, ktorý potrebujeme riešiť. V závislosti od problému budeme pracovať s viacerými reprezentáciami grafu, čo dovoľuje jeho efektívnejšie riešenie. Hlavným kritériom v hodnotení efektívnosti riešenia, je sledovať závislosť reprezentácie grafu a jeho časovú a pamäťovú zložitosť.

Matica susednosti

Matica susednosti M grafu G je štvorcová matica rádu $|V|$, kde riadky aj stĺpce matice sú indexované vrcholmi a pre prvok $a_{i,j}$ a hranu e platí,

$$a_{i,j} = \begin{cases} 1, e \in E, \\ 0, e \notin E \end{cases}$$

V neorientovanom grafe je matica susedností symetrická podľa hlavnej diagonály, tj. pre každé i, j, kde $i \neq j$ platí, že $a_{i,j} = a_{j,i}$. V orientovanom grafe matica susednosti nemusí byť symetrická. Hlavná diagonála obsahuje iba nuly, pretože obyčajný graf nemá slučky.

Pamäťový nárok na graf reprezentovaný maticou je (N^2). Overenie, či sú dva vrcholy susedné, vieme spraviť v konštantnom čase. V konštantnej časovej zložitosti vieme pridať hranu do grafu, resp. ju vymazať. Zistenie stupňa vrchola a zistenie všetkých susedov sa dá spraviť s asymptotickou časovou zložitosťou $O(|V|)$. Pre jednoduchosť reprezentácie grafu touto maticou, sme sa rozhodli, že grafy budeme generovať práve prostredníctvom nej.

Zoznam nasledovníkov

Zoznam nasledovníkov je zoznam, kde si ku každému vrchola uložíme zoznam jeho susedov. Pre neorientované grafy si každú hranu pamätáme dva-krát, pre orientované len raz. Takáto reprezentácia je pamäťovo aj časovo efektívna. Pamäťový nárok na graf reprezentovaný týmto spôsobom je práve $2|E|$.

Ak číslo d_{max} je najväčší stupeň vrchola zo všetkých vrcholov konkrétneho grafu, potom overovanie, či sú dva vrcholy susedné a prechod cez všetkých susedov vrchola môžeme uskutočniť s časovou zložitosťou najviac d_{max}. Požiadavku na pridanie hrany do zoznamu nasledovníkov vieme vykonať v konštantnom čase, kým na odstránenie hrany potrebujeme $2d_{max}$ času.

Zoznam hrán

Pri reprezentácii grafu ako zoznamu hrán, si uložíme všetky hrany ako dvojice vrcholov do jedného zoznamu. Takáto reprezentácia grafu je vhodná pre orientované aj neorientované grafy. Pamäťový nárok na zoznam hrán je $2|E|$. Overenie, či je sú 2 vrcholy

susedné, ako aj výpis všetkých susedov vrchola a vymazanie hrany z grafu pri takejto reprezentácii grafu prebieha s časovou zložitosťou $O(|E|)$. Pridanie hrany vieme aj v tejto reprezentácii vykonať v konštantnom čase.

1.2.6 Záver kapitoly

V tejto kapitole sme uviedli niekoľko vybraných štandardných definícií vlastností grafov. Každá z nich sa dá vizualizovať 2 a viac farbami. Uviedli sme a porovnali pamäťové, resp. časové zložitosti rôznych reprezentácií grafu.

Kapitola 2

Vizualizácia

2.1 Vizualizácia informácií

S nárastom množstva informácií produkovaných informačnými systémami klesá prehľadnosť dát. Dáta v súčasnosti získavame rôznymi nástrojmi, analýzou, meraniami, štatistickými metódami. Ťažko sa v nich orientuje a neustály príjem informácií si vyžaduje efektívne skúmanie, aby z nich boli extrahované len podstatné informácie.

Preto vzniká potreba tieto údaje nejakým spôsobom spracovať. V dnešnej dobe je čoraz populárnejšie spracovať dáta pomocou vizualizácie. Pojem vizualizácie môžeme chápať ako proces zobrazovania informácií do grafickej podoby.

S vizualizovanými dátami sa človek stretáva v živote na rôznych miestach, v rôznych formách. V tlači, v televízii, na internete... a pod. Výhodou vizualizovaných dát je ľahšie porozumenie ich významu, intuitívnejšia predstava a lepšia čitateľnosť. Vizualizované dáta sú teda bližšie ľudskému porozumeniu a pomocou nich získava človek intuitívnejší obraz o informáciách.

2.1.1 Typy vizualizácií

Vizualizovať, teda priradiť alfanumerickým dátam vizuálnu reprezentáciu, môžeme v súčasnosti mnoho vecí. V závislosti od podnetu vizualizácie rozlišujeme rôzne druhy vizualizácií.

Výsledkom statickej vizualizácie je statický objekt, kompletne vymodelovaná perspektíva, napr. obraz, mapa, ktorý podáva informáciu o dátach s vedomím, že tento

výsledok sa nebude meniť. Pri dynamickej vizualizácii získavame predstavu ako sa vizualizované dáta v závislosti od času menia, získavame vymodelovanú scénu. Príkladom takejto vizualizácie je animácia, video, film.

Interaktívna vizualizácia umožňuje používateľovi zasahovať do výsledku a na jeho podnety zobrazovať nové výsledky. Pod takouto vizualizáciou si môžeme predstaviť interaktívnu mapu, pohyb používateľa po scéne v reálnom čase vo vymedzenom priestore [14].

Výsledkom programu tejto bakalárskej práce bude práve statická vizualizácia s možnosťou interaktívneho pohybu po 2D scéne, v ktorej budú zobrazené grafy, ktoré farbíme podľa toho, či majú nami vybrané vlastnosti. Klikaním na smerové tlačítka bude možné pohybovať sa po vykresľovanej proche.

2.1.2 Proces vizualizácie

Proces vizualizácie podľa [15] tvoria dve hlavné fázy, rozdelené do siedmich krokov. par *Získanie dát*: Využívanie dát zo zdroja, ktorým môže byť databáza, súbor, web, server. Dáta môžeme získať meraním, analýzou, štatistickými výpočtami,...

Triedenie dát: Rozdelenie dát do kategórií, návrh vhodnej dátovej štruktúry, konverzia dátových typov,...

Filtrovanie dát: Chceme pracovať len s dátami, ktoré pre nás majú informačnú hodnotu. Teda odstránime nadbytočné a nesúvisiace dáta. Patrí sem normalizácia dát, úprava viacerých typov na spoločný tvar.

Dolovanie znalostí: Hľadanie vzoriek (pattern) v dátach, hľadanie matematického kontextu, modelu. Využívanie matematickej štatistiky a hĺbkovej analýzy.

Prezentácia: Potreba vhodnej reprezentácie dát.

Zjemňovanie: Zlepšovanie grafického výstupu, pridávanie detailov.

Interakcia: Pridanie možnosti manipulácie s vizualizáciou. Pridanie možností pre používateľa, aby mohol s výstupom ľubovoľne pracovať.

Takáto vizualizácia je interaktívna. Teda jednotlivé kroky sú medzi sebou prepojené a nenasledujú vždy nutne za sebou v poradí, tak ako je uvedené vyššie, ale platí tu možnosť vrátenia sa k predošlým bodom procesu.

2.1.3 Zobrazenie grafov

Grafy budeme v tejto práci značiť (marker) ako body v rovine. Každý graf bude reprezentovaný štvorcom konštantných rozmerov. Farba štvorca bude zodpovedať farbe na stupnici odtieňu v závislosti od vlastností, ktoré boli na grafe skúmané a ktoré konkrétny graf má.

V práci budeme zobrazovať aj jednotlivé grafy. Konkrétny graf bude zobrazený v rovine v samostatnom pracovnom okne. Vrcholy grafu budú štvorce konštantnej veľkosti, a hrany grafu budu reprezentované priamymi čiarami spájajúcimi *incidentné* vrcholy (Veta 1.1.1.5).

2.2 Zobrazenie grafov do roviny

Grafy budú zobrazované do roviny postupne, v závislosti od toho, ako ich budeme postupne generovať, analyzovať a farbiť.

Vzdialenosť grafu od stredu súradnicovej sústavy bude určená počtom vrcholov grafu, čím dosiahneme to, že všetky grafy o rovnakom počte vrcholov budú zobrazené na kružnici. Najprv budú zobrazené grafy bez hrán. Grafy budú generované lexikograficky pomocou matice susednosti, čím zabezpečíme, že budú vygenerované všetky grafy postupne v poradí. Najprv sa zobrazia grafy s jednou hranou, potom dvomi hranami, atď.

Dopredu si vypočítame presný počet grafov, čo zabezpečí, že grafy budú na kružnicu zobrazené rovnomerne. Zobrazovanie začneme v Gaussovej rovine na osi x a grafy budeme postupne pridávať v kladnom uhlovom smere, teda "proti smeru chodu hodinových ručičiek".

2.3 Farbenie

Každý graf bude analyzovaný a ohodnotený na vybranom počte vlastností. V závislosti od ohodnotenia grafu, bude mu pridelený odtieň farby, ktorú si používateľ nastaví v nastavení programu. Odtieň je rovnomerne škálovaný medzi odtieňmi bielej farby a farbou používateľom vybranej, vo východzom nastavení čiernej farby.

Kapitola 3

Implementácia

V tejto kapitole uvedieme, ako budú jednotlivé časti práce naimplementované, vysvetlíme čo budeme robiť, a ukážeme časti zdrojového kódu s komentárom a vysvetlením.

3.1 Špecifikácia

Cieľom práce je zostrojiť nástroj, ktorý dokáže vygenerovať grafy, zobraziť ich v rovine a ofarbiť ich podľa toho, ktoré zo skúmaných vlastností majú.

Nástroj bude určený ako pre laickú verejnosť, tak aj na akademické a výskumné účely. Preto sa s ohľadom na používateľa budeme pri implementácii držať intuitívnosti.

Predpokladá sa, že tento nástroj bude v budúcnosti upravovaný a jeho funkcionalita bude rozširovaná podľa potrieb používateľa. Preto sa budeme snažiť o jednoduchosť, aby sme zachovali prehľadnosť a ľahšiu ďalšiu úpravu.

3.1.1 Štruktúra aplikácie

Proces tvorby výstupu je rozdelený do troch častí. Generovanie grafov, analýza grafu a jeho zobrazenie. Každú časť bližšie popíšeme v samostatnej kapitole.

Dôležité časti zdrojového kódu budú umiestnené v samostatných súboroch, pre zvýšenie prehľadnosti.

3.1.2 Výber programovacieho jazyka

Výber programovacieho jazyka závisí od niekoľkých kritérií.

Z charakteru problému a toho, že budeme pracovať s grafmi ako abstraktnou dátovou štruktúrou je vhodné využiť programovací jazyk, ktorý nám čo najviac uľahčí prácu. Preto sme nevybrali nízko-úrovňový jazyk, ale budeme využívať výhody objektovo orientovaného programovacieho jazyka.

Vzhľadom na cieľovú skupinu používateľov, a dodržanie intuitívnosti aj pre laickú verejnosť, vybrali sme aplikáciu s používateľským rozhraním. Teda aplikácia nebude spustená v konzole, ale bude fungovať v jednom alebo viacerým oknách. Bude ju možné ovládať základnými vstupnými zariadeniami, napr. myš, klávesnica.

Aj základná konštrukcia aplikácie pracuje s väčším množstvom grafov, analyzuje algoritmicky náročnejšie vlastnosti a vzhľadom k akademickým prístupom a výskumným cieľom sa neočakáva výpočet v reálnom čase. Taktiež výpočet bude prebiehať na stroji s vyššou výpočtovou silou.

Ďalším kritériom výberu bola multiplatformovosť. Aplikáciu by sme mali byť schopní spustiť na ľubovolnom operačnom systéme. Keďže hovoríme o desktopovej aplikácií, potrebujeme vedieť spustiť aplikáciu na stroji s operačným systémom Microsoft Windows a taktiež na operačnom systéme s založeným na jadre linuxu.
Program bol testovaný na Windows 7 a Ubuntu 14.07.

Tieto kritéria nám zúžili výber, ale programovacích jazykov spĺňajúcich dané kritérium je stále dostatočne veľa. Do užšieho výberu sa dostali jazyky C++, Python, Java, C Sharp.

Po zhodnotení všetkých týchto jazykov sme vybrali práve C Sharp, ktorý na platforme Microsoft .NET ponúka na tvorbu desktopových používateľských prostredí hneď dva podsystémy. Windows Forms (WF) a Windows Presentation Foundation (WPF). Obe tieto grafické podsystémy majú svoje klady a zápory v závislosti od charakteru zadania, na ktoré ich chceme použiť.

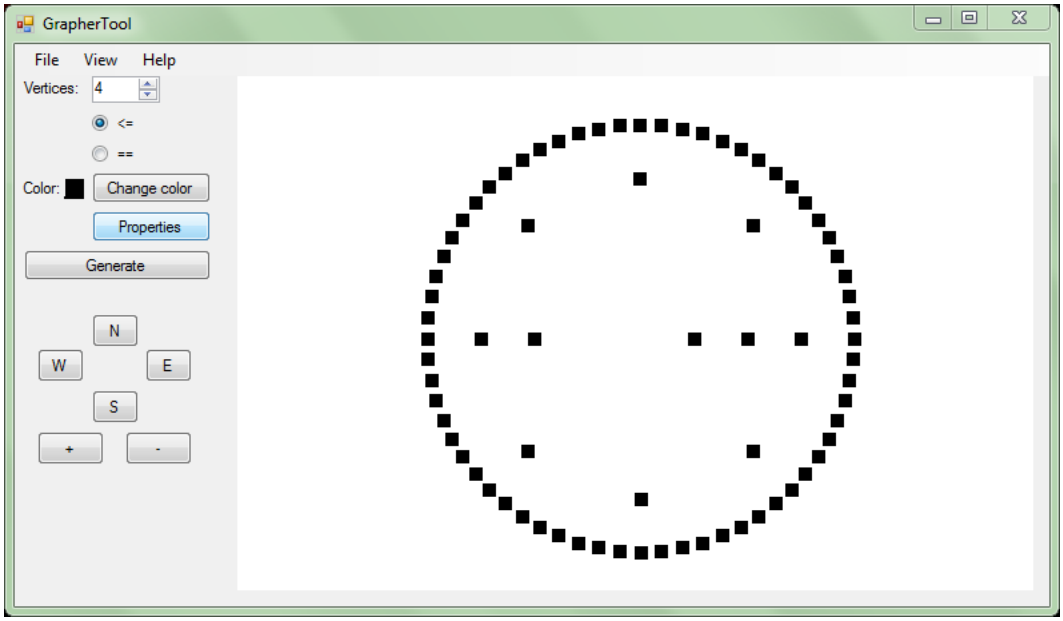
My sme vybrali Windows Forms, pre jednoduchosť implementácie a pre podporu práve takýchto aplikácií [16]. Celá implementácia bude písaná tak, aby mohol schopnejší programátor pomerne dosť jednoducho prepísať zdrojové súbory do ľubovoľného objektovo orientovaného jazyka.

3.1.3 Vzhľad

Aplikácia bude obsahovať dve hlavné časti, reprezentované dvomi oknami.

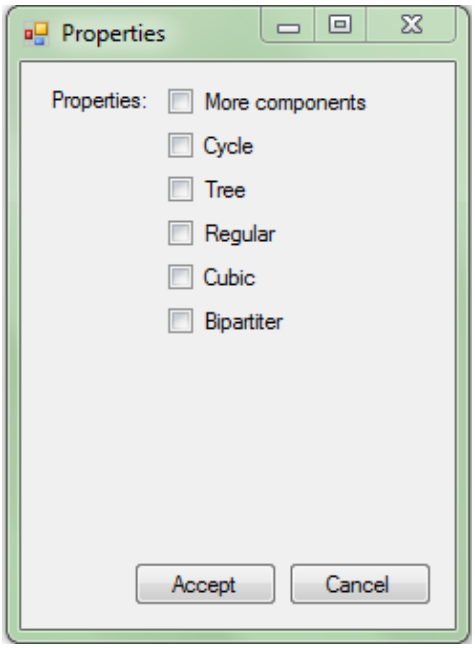
Prvé okno bude slúžiť používateľovi na vyber potrebných nastavení pre ďalší výpočet (Obrázok 3.1). Používateľ si bude môcť zvoliť, či chce generovať grafy o vybranom počte vrcholov, alebo či chce do výpočtu zahrnúť všetky grafy s menším počtom vrcholov. Používateľ si bude môcť vybrať farbu, ktorej odtiene budu zobrazené vo vizualizácii. Časť okna bude reprezentovať scénu, tj. 2D rovinu, v ktorej budú zobrazené štvorce reprezentujúce grafy.

Používateľ bude mať možnosť pohybu po scéne pomocou smerovacích tlačidiel a tlačidiel na priblíženie, resp. oddialenie scény. Po kliknutí na graf (štvorec reprezentujúci graf), vytvoríme nové okno v ktorom bude vizualizovaný konkrétny graf.



Obr. 3.1: Ukážka pracovného prostredia systému

Používateľovi sa po kliknutí na tlačidlo *Properties* otvorí okno s ponukou zobrazujúcou, ktoré konkrétne vlastnosti, z výberu naimplementovaných, chce používateľ práve analyzovať (Obrázok 3.2).



Obr. 3.2: Ukážka výberu vlastností grafu

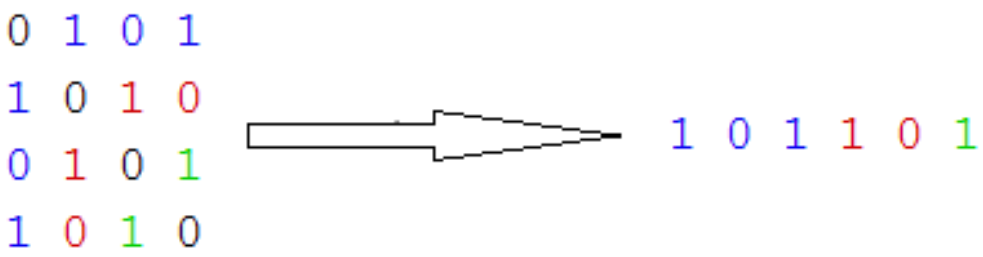
Po kliknutí na tlačidlo *Accept*, bude výber vlastností uložený do objektu *Analyser*, a následne sa budú na grafe analyzovať len vybrané vlastnosti.

3.2 Generovanie grafov

Každý graf bude reprezentovaný objektom typu *Graph*. Grafy budú generované lexikograficky pomocou matice susednosti. Keďže sa práca venuje neorientovaným grafom, bez ujmy na všeobecnosti, stačí brať do úvahy buď hornú alebo spodnú časť matice rozdelenej diagonálou. Pre jednoduchosť implementácie uvažujme len o hornej časti a spodnú časť si podľa nej dopočítame.

O tejto časti matice budeme pre jednoduchosť implementácie uvažovať ako o poli prvkov (Obrázok 3.3). Ak n je počet vrcholov grafu, potom pre tento graf platí, že má $(n-1)(n-2)/2$ maximálny počet hrán, to bude zároveň veľkosť poľa, ktorú budeme potrebovať na vygenerovanie grafov.

Do tohto poľa budeme lexikograficky generovať binárne hodnoty, tj. nuly a jednotky, čím nám po prečítaní celého poľa vznikne jeden binárny reťazec reprezentujúci jednu konfiguráciu grafu, tj. rozmiestnenie hrán medzi vrcholmi.



Obr. 3.3: Transformácia matice susednosti do poľa

Na generovanie konfigurácií využijeme triedu *GraphGenerator.cs*, ktorá bude mať implementovanú metódu *getGraphs(n)*. Táto metóda vracia zoznam všetkých grafov s n vrcholmi v lexikografickom usporiadaní. V tejto metóde si do zoznamu vygenerujeme binárne čísla, ktoré konvertujeme na matice susedností. Vložením takejto matice do konštruktora nového objektu *Graph* vytvárame nový graf, ktorý si uložíme do zoznamu grafov.

```
1  public ArrayList getGraphs(int n)
2  {
3      ArrayList graphs = new ArrayList(); // list of objects <Graph>
4      if (n == 1) // create simple graph with one vertex
5      {
6          int[,] a = new int[1, 1];
7          a[0, 0] = 0;
8          graphs.Add(new Graph(a, 1, id++));
9      } else if (n > 1) // create graph with n vertices
10     {
11         // list of bin strings that represent graphs
12         ArrayList binStrings = getList(n);
13         /*
14         * every binary string we need to convert into matrix and then
15         * make Graph from it
16         */
17         foreach (string s in binStrings)
18         {
19             //convert array to matrix
20             int[,] tempArray = stringToArray2D(s, n);
21             // create new graph with unique id
22             Graph tempGraph = new Graph(tempArray, n, id++);
23             graphs.Add(tempGraph); // add new graph into list
24             //tempGraph.print(); // print graph in to console
25         }
26     }
27     return graphs;
28 }
```

Metóda *getList(n)* vráti zoznam binárnych reťazcov v lexikografickom poradí, napr. pre dĺžku 3: 000, 001, 010, 100, 011, 101, 110, 111. Tieto reťazce bude potrebné konvertovať na matice. To bude úlohou metódy *stringToArray2D(s,n)*, ktorej argumentom s bude binárny reťazec a n bude počet vrcholov grafu.

```
1  private int[,] stringToArray2D(string s, int nVertices)
2  {
3      int right = nVertices - 1;
4      if (s.Length == (right * (right + 1)/2))
5          // if numbers are correct, return
6      {
7          int index = 0;
8          int[,] res = new int[nVertices, nVertices];
9          // fill array with 0 or right value
10         for (int i = 0; i < nVertices; i++)
11         {
12             for (int j = 0; j < nVertices; j++)
13             {
14                 res[i, j] = 0;
15                 res[i, j] = (int)Char.GetNumericValue(s[index]);
16                 res[j, i] = res[i, j];
17                 index++;
18             }
19         }
20         return res;
21     }
22     return null;
23 }
```

Tieto metódy budú použité pre každý počet vrcholov presný, vopred vypočítaný počet krát, preto sa môžeme spoľahnúť, že sa nám generovanie grafov v žiadnom prípade nezacyklí alebo inak nezlyhá, teda skončí v konečnom čase po konečnom počte krokov. Obmedzený budeme len pamäťovými prostriedkami a maximálnou hodnotou dátového typu integer, čo je hodnota 2^{31} bitov.

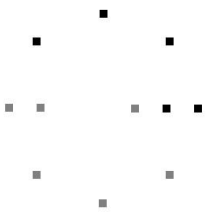
3.3 Analýza grafov

Pre každý graf zo zoznamu grafov je potrebné spustiť analýzu jeho vlastností. Zisťovanie vlastností grafov bude zabezpečovať trieda *Analyser*, a jej metóda Analyse(Graph g).

Táto metóda dostane v argumente objekt Graph reprezentujúci jeden konkrétny graf. Následne sa v metóde zavolajú jednotlivé testy pre určovanie konkrétnych vlastností.

3.3.1 Komponenty súvislosti

Bude nás zaujímať, či má graf jeden alebo viac komponentov súvislosti (Obrázok 3.4). Počet komponentov grafu budeme zisťovať pomocou prehľadávania do hĺbky. (Veta 1.2.1.2)Na začiatku budú inicializované všetky vrcholy grafu ako nenavštívené. Následne pre prvý vrchol v zistíme, či bol navštívený. Ak nebol navštívený, spustíme prehľadávanie do hĺbky, tj. zavoláme metódu *DFS(v)*. Metóda DFS bude pracovať tak, že vrchol v označí ako navštívený a pre každý vrchol u susediaci s vrcholom v zistí, či bol vrchol u navštívený. Ak u nebol navštívený, tak rekurzívne zavoláme metódu DFS(u) s argumentom u. Po skončení prehľadávania do hĺbky skontrolujeme, či sú všetky vrcholy označené ako navštívené. Ak sú, potom budeme vedieť nájsť cestu z jedného vrcholu do všetkých ostatných, a teda všetky vrcholy sú v jednom komponente súvislosti, čiže graf je súvislý (Definícia 1.2.1.1). Ak nájdeme aspoň jeden neoznačený vrchol, potom má graf viac komponentov.



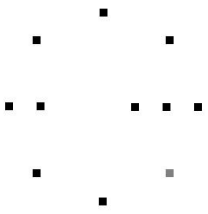
Obr. 3.4: Súvislé grafy sú zobrazené sivou farbou

3.3.2 Cyklus

Hľadanie, či graf obsahuje aspoň jeden cyklus 1.1.3.4 budeme robiť podobne ako pri hľadaní komponentov, pomocou prehľadávania do hĺbky. (Obrázok 3.5)

Na začiatku budú inicializované všetky vrcholy grafu ako nenavštívené. Následne pre každý vrchol v, ktorý ešte nebol navštívený spustíme metódu DFS(v). Metóda DFS označí vrchol v ako navštívený a rekurzívne spustí metódu DFS(u), kde u je každý vrchol susediaci s v. Rozdiel v algoritme oproti hľadaniu komponentov bude ten, že pokiaľ DFS nájde počas prehľadávania do hĺbky nejaký susedný vrchol, ktorý už bude v čase volania funkcie navštívený, skončí svoju činnosť, pretože našiel cyklus.

Tieto dva algoritmy sa dajú upraviť a spojiť do jedného prehľadávania do hĺbky, ale pre jednoduchosť a prehľadnosť ich naimplementujeme oddelene.

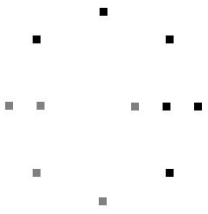


Obr. 3.5: Grafy obsahujúce cyklus sú znázornené sivou farbou

3.3.3 Strom

Na to, aby bol graf stromom, musí spĺňať niekoľko podmienok (Veta 1.2.2.2). To, či bude graf obsahovať viac komponentov súvislosti, či cyklus už budeme vedieť určiť z vyššie implementovaných vlastností. Stačí nám teda overiť, či bude obsahovať $n - 1$ hrán, pre počet vrcholov grafu n. Tento počet získame prehľadaním matice susednosti a postupných spočítaním jednotiek reprezentujúcich hrany. (Obrázok 3.6)

Budeme sa pozerať len do hornej časti matice rozdelenej diagonálou. Ak zistíme, že graf je acyklický (Veta 1.1.3.5), že má nemá viac komponentov súvislosti (Veta 1.2.1.2), a že obsahuje práve $n - 1$ hrán, potom budeme môcť tvrdiť, že tento graf je *stromom*.

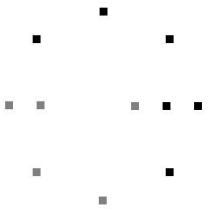


Obr. 3.6: Grafy, ktoré sú stromom sú znázornené sivou farbou

3.3.4 K-regulárnosť grafu

Analýza vlastnosti k-regulárnosti (Veta 1.1.2.5) grafu bude prebiehať tak, že postupne pre každý vrchol budeme spočítavať počet jeho susedov, teda hrán incidentných s týmto vrcholom (Veta 1.1.1.5). Ak budú všetky vrcholy rovnakého stupňa (Veta 1.1.2.2), po-

tom graf bude k-regulárny, kde k je stupeňa vrchola (Obrázok 3.7). Ak počas prehľadávania zistíme, že niektorý vrchol ma iný počet susedných vrcholov ako prvý vrchol, potom svoju činnosť hľadania tejto vlastnosti budeme môcť ukončiť s vedomím, že graf už nemôže byť k-regulárny.



Obr. 3.7: Grafy, ktoré sú k-regulárne sú znázornené sivou farbou

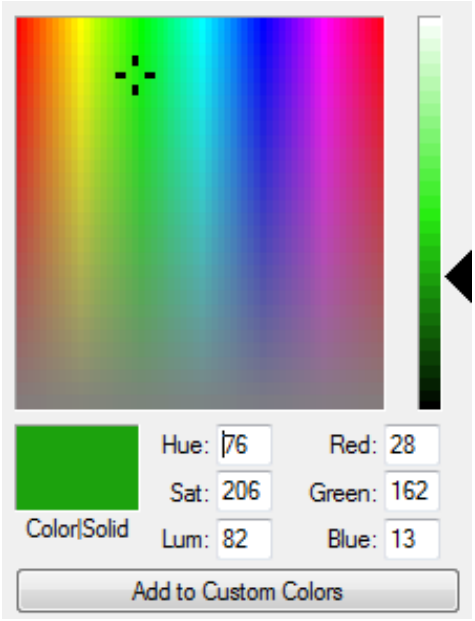
3.3.5 Kubický graf

Určovanie, či bude graf kubickým grafom (Veta 1.2.4.1) budeme robiť súčasne s hľadaním k-regulárnosti, pretože ide o zhodný algoritmus, s takým rozšírením, že si budeme pamätať aj stupeň vrcholov. Ak po analýze grafu zistíme, že bude graf k-regulárny, kde stupeň vrcholov je práve 3, potom bude tento graf kubický.

3.4 Farbenie grafu

Každému grafu bude podľa jeho vlastností určená farba. Používateľ si môže vybrať ľubovolnú farbu, ktorej odtieňmi budú vlastnosti grafov reprezentované. Pre lepšiu vizualizáciu a obmedzenie nevhodnej farby budeme postupovať nasledovne.

Vybratej farbe v modely RGB (Obrázok 3.8) priradíme zodpovedajúcu farbu v modely HSL. V tomto modeli nastavíme konkrétnej farbe sýtosť na maximálnu hodnotu, tj. 255. Výsledkom bude nová farba, sýtejšia oproti pôvodnej vybratej. V ďalšom kroku rovnomerne rozdelíme odtiene farby, čo bude prebiehať tak, že maximálnu hodnotu svietivosti budeme deliť počtom odtieňov. Každej vlastnosti, resp. kombinácií vlastností budeme postupne priraďovať jeden odtieň. V tejto časti práce je nutný zásah používateľa, pretože nástroj nebude schopný sám dokázať detegovať a určiť, ktoré vlastnosti spolu súvisia, a teda nebude môcť vedieť efektívne zvoliť správny potrebný počet odtieňov.



Obr. 3.8: Výber farby, model RGB a HSL

3.5 Mapovanie grafu do roviny a zobrazenie

Grafy budeme zobrazovať ako štvorce na kružniciach s polomerom rovných počtu vrcholov. Zobrazené budú do 2D roviny s počiatkom súradníc v strede zobrazovacieho panela. Každému grafu priradíme jeho súradnice začiatočnej polohy v rovine.

Vypočítame si počet všetkých grafov s rovnakým počtom vrcholov, ozn. z. Následne si určíme uhlový posun *theta*, tj. hodnotu uhla, o ktorú sa budú grafy medzi sebou líšiť. V rovnici 3.1 premenná i označuje poradie vrchola na kružnici.

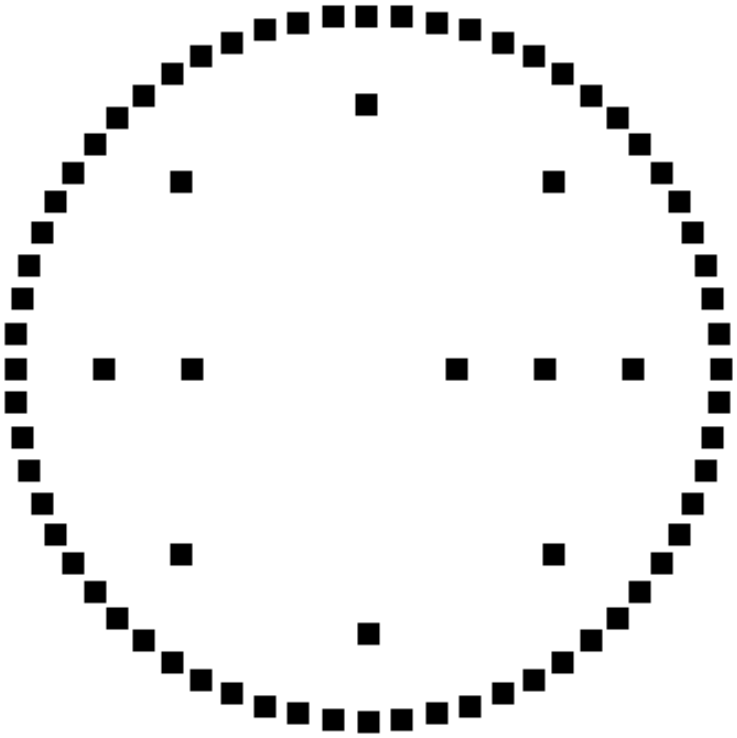
$$theta = (-2) * PI * \frac{i}{z} \tag{3.1}$$

Potom budeme môcť vyčísliť súradnice na osi x a y (Rovnice 3.2 a 3.3). V týchto rovniciach $vertexCount$ označuje počet vrcholov a *const* je konštanta, zvolená tak, aby grafy boli vzdialené od seba dostatočne ďaleko.

$$x = vertexCount * const * cos(theta) \tag{3.2}$$

$$y = vertexCount * const * sin(theta) \tag{3.3}$$

Pri takomto mapovaní, budú grafy bez hrán zobrazené do roviny na oxi x a následne s nárastom počtu hrán a ich lexikografickým usporiadaním budú grafy zobrazené na kružnici v kladnom smere, tj. proti smeru chodu hodinových ručičiek (Obrázok 3.9).



Obr. 3.9: Ukážka mapovania grafov s počtom vrcholov 1 až 4

Pre vysoký počet grafov a neprehľadnosť ich zobrazenia v rovine bude implementovaný nástroj na pohyb v rovine. Používateľ pomocou smerových tlačidiel a tlačidiel pre priblíženie, resp. oddialenie zobrazovanej scény, bude môcť pohybovať grafmi v rovine. Súradnice grafu budú implementované pomocou matice, pohyb v rovine a priblíženie, resp. oddialenie scény bude realizované prostredníctvom násobenia tejto matice vhodnou maticou pre transformáciu.

3.6 Časová zložitosť výpočtu

V implementácii tejto práce budeme dbať na jednoduchosť, nie efektívnosť, dôsledkom čoho bude pomalšia práca programu.

Generovanie grafov. Generovanie binárnych reťazcov, z ktorých skladáme matice prebieha v časovej zložitosti $O(2^n)$, kde n je dĺžka reťazca. Následne tieto reťazce kon-

vertuje do poľa s časovou zložitosťou $O(|V|^2)$.

Analýza grafu. Pri hľadaní komponentov a cyklov grafu budeme využívať prehľadávanie do hĺbky. Algoritmus prehľadávania do hĺbky pracuje s časovou zložitosťou $O(|V| + |E|)$. Hľadanie k-regulárnosti, resp. kubickosti grafu bude prebiehať závislosti od reprezentácie grafu. My budeme používať maticu susednosti, teda analýza týchto vlastností bude prebiehať s časovou zložitosťou $O(|V|^2)$. Analýza vlastnosti, či je graf stromom, bude určená z predchádzajúcich vlastností a z lineárneho prehľadania všetkých hrán, teda časová zložitosť bude $O(|E|)$.

Určovanie farby a súradníc grafu prebieha v konštantnom čase $O(1)$.

Záver

V tejto práci sme sa venovali navrhnutiu a implementácii nástroja, pomocou ktorého používateľ bude môcť vizualizovať vybrané vlastnosti grafov. Výstupy z tejto aplikácie dokazujú, že cieľ práce bol splnený. (Obrázky 3.4, 3.5, 3.6, 3.7)

Aplikácia slúži ako základný nástroj pre generovanie a zobrazovanie neorientovaných grafov v rovine. Bola naprogramovaná v programovacom jazyku C Sharp, v grafickom podsystéme Windows Forms. Počas programovania sme sa snažili klásť dôraz na jednoduchosť, čo by malo v budúcnosti dopomôcť k ľahšej úprave či rozšíreniu aplikácie.

S nárastom počtu vrcholov, exponenciálne stúpa počet grafov (Tabuľka 3.1). Počet grafov s n vrcholmi môžeme vyčísliť ako $2^{\binom{n}{2}}$. V predchádzajúcej práci [8] sa predchodca dostal po počet vrcholov 4. Môžeme prehlásiť, že túto hranicu sme prekonali, dokážeme pohodlne pracovať s grafmi do počtu vrcholov 6. Pre vyšší počet vrcholov máme pri súčasných testovaniach problém s nedostatkom výpočtovej pamäte. V budúcnosti môžeme o výpočty požiadať niektoré super-počítačové centrum.

Počet vrcholov n	$\binom{n}{2}$	**Počet grafov s** n **vrcholmi**
1	0	1
2	1	2
3	3	8
4	6	64
5	10	1 024
6	15	32 768
7	21	2 097 152
8	28	268 435 456

Tabuľka 3.1: Závislosť počtu grafov od počtu vrcholov

Na každom z vygenerovaných grafov analyzujeme 5 vybraných vlastností: či je graf súvislý, či obsahuje cyklus, či je graf stromom, či je graf k-regulárny a či je kubický. Vybrali sme jednoduchšie vlastnosti pre demonštrovanie funkcionality nástroja.

Táto aplikácia má v sebe potenciál byť lepším nástrojom, je otvorená možnostiam a

schopnejší programátor by nemal mať problém pokračovať v tejto práci. V súvislosti s rozširovaním môžeme spomenúť, že nástroj by v budúcnosti mohol dokázať efektívnejšie analyzovať vlastnosti za predpokladu, že bude vymyslené a implementované určovanie izomorfných grafov.

Ďalším vylepšením by mala byť implementácia analýzy nových vlastností grafov, či výhodné ukladanie dát do databázy, na server,... a pod. Zaujímavým no zložitým vylepšením by bola implementácia a urýchlenie výpočtu práce pomocou viacerých vláken.

Aplikácia poskytuje do budúcnosti ľahké zavedenie orientovaných grafov, ohodnotených hrán, multihrán, a to vďaka generovaniu a reprezentácii grafu prostredníctvom matice susednosti. Aplikácia je v neustálom procese vývoja.

Príloha práce je v podobe CD. Obsahuje zdrojové kódy a spustiteľnú verziu aplikácie vo formáte .exe.

Literatúra

[1] Jerrold W Grossman. The evolution of the mathematical research collaboration graph. *Congressus Numerantium*, pages 201–212, 2002.

[2] Leonhard Euler. Leonhard Euler and the Königsberg bridges. *Scientific American*, 189(1):66–70, 1953.

[3] Elmar Teufl and Stephan Wagner. Spanning trees of finite Sierpiński graphs. *DMTCS Proceedings*, (1), 2006.

[4] Eric M Rains and Neil JA Sloane. On cayley's enumeration of alkanes (or 4-valent trees). *J. Integer Sequences*, 2, 1999.

[5] Stephan Mertens. Computational complexity for physicists. *Computing in Science & Engineering*, 4(3):31–47, 2002.

[6] Neil Robertson, Daniel Sanders, Paul Seymour, and Robin Thomas. The four-colour theorem. *journal of combinatorial theory, Series B*, 70(1):2–44, 1997.

[7] Jaroslav Nešetřil, Eva Milková, and Helena Nešetřilová. Otakar Borůvka on minimum spanning tree problem translation of both the 1926 papers, comments, history. *Discrete Mathematics*, 233(1):3–36, 2001.

[8] Ernest Štibrányi. Vizualizácia teórie grafov. Diplomová práca, Univerzita Komenského v Bratislave, 1997.

[9] Carolyn McCreary and Larry Barowski. Vgj: Visualizing graphs through java. In *Graph Drawing*, pages 454–455. Springer, 1998.

[10] Pavol Korínek. Vizualizácia vybraných algoritmov a vlastnosti z teórie grafov. Diplomová práca, Univerzita Komenského v Bratislave, 2006.

[11] Ladislav Unčovský, Ladislav Kovaľ, Peter Fiala, Stanislav Vejmola, and Bruno Musil. *Modely sieťovej analýzy*. Alfa, 1991.

[12] Reinhard Diestel. *Graph theory {graduate texts in mathematics; 173}*. Springer-Verlag Berlin and Heidelberg GmbH & amp, 2000.

[13] Ondrej Holotňák. Zbierka úloh z teórie grafov. Diplomová práca, Univerzita Komenského v Bratislave, 2006.

[14] Ben Shneiderman Benjamin B. Bederson. *The Craft of Information Visualization: Readings and Reflections*. Morgan Kaufmann Publishers, 2003.

[15] Ben Fry. *Visualizing data: Exploring and explaining data with the processing environment*. O'Reilly Media, Inc., 2007.

[16] Joe Mayo. *Microsoft Visual Studio A Beginner's Guide*. The McGraw-Hill Companies, 2010.

Dodatok A

Elektronická verzia práce na CD

Disk CD obsahuje prácu v elektronickej podobe, zahŕňa zdrojové kódy a spustiteľnú verziu aplikácie pod systémom Windows. Obsahuje jednoduchú dokumentáciu. Štruktúra: priečinok projektu, bakalárska práca vo formáte .pdf, .tex