Univerzita Komenského v Bratislave

Fakulta matematiky, fyziky a informatiky

Niektoré vlastnosti signovaných grafov

Bakalárska práca

2013

Kamil Bulko

Univerzita Komenského v Bratislave

Fakulta matematiky, fyziky a informatiky

Niektoré vlastnosti signovaných grafov

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra Informatiky
Školiteľ:	RNDr. Edita Rollová, PhD.

Bratislava, 2013

Kamil Bulko

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Kamil Bulko
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský

Názov: Niektoré vlastnosti signovaných grafov

Cieľ: Navrhnúť a naprogramovať efektívny algoritmus, ktorý rozhodne, či sú dané dva signované grafy izomorfné.

Vedúci: RNDr. Edita Rollová, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: doc. RNDr. Daniel Olejár, PhD.

Dátum zadania: 25.10.2011

Dátum schválenia: 25.10.2011 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestné prehlásenie

Čestne prehlasujem, že prácu som vypracoval sám s pomocou uvedenej literatúry.

Bratislava, 01.06.2013 Podpis .....

Poďakovanie

Ďakujem vedúcej bakalárskej práce Edite Rollovej za cenné rady a pripomienky, bez ktorých by táto práca asi nevznikla, blízkym priateľom a rodine za morálnu podporu.

Kamil Bulko

Abstrakt

V tejto práci sa zameriavame na špeciálnu nadtriedu grafov nazývaných signované grafy. Ide o grafy s kladnými a zápornými znamienkami na hranách. Študujeme rozdiely medzi nimi, skúmame kedy sú dva signované grafy ekvivalentné a kedy izomorfné. Napokon navrhujeme a rozoberáme algoritmus, ktorý hľadá izomorfizmus daných signovaných grafov.

Kľúčové slová: graf, signovaný graf, izomorfizmus, algoritmus

Abstract

In this paper, we focus on a special superclass of the graphs called signed graphs. Signed graphs are graphs with positive and negative signs on edges. We study the differences between them, we investigate when the two signed graphs are equivalent or isomorphic. Finally we discuss and propose an algorithm that is looking for isomorphism of the signed graphs.

Keywords: graph, signed graph, isomorphism, algorithm

Zoznam obrázkov

Obsah

Úvod

Pre riešenie problému z reality je vhodné vytvoriť si istý teoretický model, ktorý prirodzene reprezentuje daný problém. Tak ako poznáme vo fyzike matematický aparát vektorov, tak v informatike máme silný aparát teóriu grafov. V tejto práci sa budeme zaoberať zaujímavým typom grafov, takzvanými signovanými grafmi. Signované grafy sú vlastne grafy so znamienkami na hranách a grafu zodpovedá celo pozitívny signovaný graf. Signované grafy študujeme, pretože sa objavujú v rôznych oblastiach, napríklad v oblasti psychológie [8], kde sú použité na modelovanie kladných a záporných vzťahov v sociálnych sieťach. Práca je členená na úvod, päť kapitol a záver.

V prvej kapitole sa nachádzajú bežné definície z teórie grafov, pojmy ktoré sú najviac potrebné pri definícii a ďalej práci so signovanými grafmi. Väčšina vecí v teórií grafov sa dá definovať viacerými spôsobmi, snahou je podať tieto definície čo najjednoduchšie.

Druhá kapitola sa venuje reprezentácií grafov v počítači. Ak chceme pracovať so signovanými grafmi, je k tomu potrebné vedieť konkrétnym spôsobom pracovať s jednoduchými grafmi zakódovanými v počítači a s tým súvisiacimi dátovými štruktúrami.

V tretej kapitole študujeme podobnosť dvoch signovaných grafov, konkrétne ekvivalenciu a izomorfizmus. Zadefinujeme izomorfizmus jednoduchých grafov a izomorfizmus signovaných grafov. Problém izomorfizmu jednoduchých grafov je výpočtový problém. Úlohou je určiť, či sú dva konečné grafy izomorfné. Nie je známe, či je to P, alebo NP-úplny problém alebo ani ani (pre viac detailov pozri [9]). V tejto kapitole prejdeme postupne k existujúcim algoritmom a podáme stručný prehľad niektorých známych algoritmov na nájdenie izomorfizmu jednoduchých grafov.

Cieľom práce je navrhnúť algoritmus, ktorý rozhodne, či sú zadané dva signované grafy izomorfné a preto v štvrtej kapitole uvedieme vlastný návrh niektorých algoritmov pre signované grafy, ich implementáciu a časovú zložitosť. V piatej kapitole ukážeme ako bonus grafickú aplikáciu pre signované grafy, v ktorej okrem iného nájdeme funkciu, ktorá vypočíta či sú alebo nie sú graficky zadané dva signované grafy izomorfné.

Kapitola 1

1 Grafy

V úvode tejto kapitoly si uveďme niektoré fakty z histórie teórie grafov. Teória grafov ako samostatná matematická disciplína vznikla v prvej polovici dvadsiateho storočia. Za prvotnú prácu teórie grafov sa považuje problém siedmych mostov mesta Kaliningrad [3], ktorú v r. 1736 vyriešil geniálny matematik Leonhard Euler. O približne 100 rokov neskôr R. Kirchoff navrhol riešenie zložitého elektrického obvodu s využitím jeho podschémy, dnes nazývanú aj kostra grafu. V roku 1859 študoval Írsky matematik R. W. Hamilton problémy „cestovania“ po vrcholoch a hranách pravidelného dvanásťstenu. Jednou z úloh, ktoré formuloval, bola aj úloha nájdenia okružnej cesty, ktorá každý vrchol dvanásťstenu obsahuje práve raz. Táto úloha sa stala predchodcom známeho problému obchodného cestujúceho. V roku 1874 Cayley pri štúdiu štrukturálnych chemických vzorcov používal grafické zobrazenie a v tejto súvislosti Sylvester v roku 1878 prvýkrát použil termín graf v dnešnom zmysle teórie grafov.

Definitívny vznik modernej teórie grafov sa viaže na rok 1936, kedy maďarský matematik D. König publikoval prvú monografiu z teórie grafov. Odvtedy sa teória grafov rozvíja v dvoch smeroch - teoretickom a algoritmickom. V tejto práci sa budeme zaoberať aj teoretickým aj algoritmickým smerom.

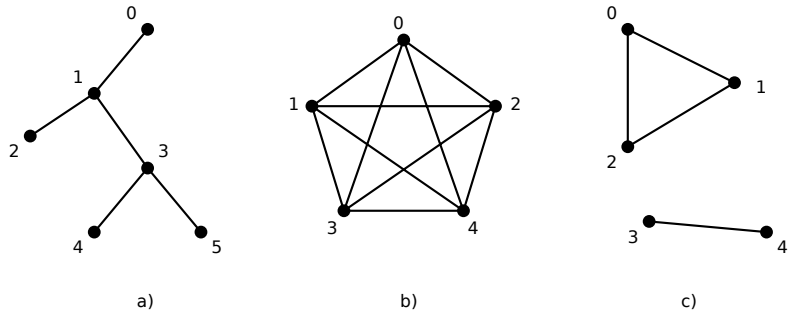
V rámci algoritmického smeru existujú „dobré“ - polynomiálne algoritmy a algoritmy ostatné - nepolynomiálne. V tejto práci sa pri hľadaní algoritmu, ktorý nájde izomorfizmy jednoduchých grafov stretneme s existujúcim algoritmom s exponenciálnou časovou zložitosťou. V teoretickom rámci nie je možné uviesť algoritmus bez rozboru jeho zložitosti, preto aj v tejto práci na záver odhadneme časovú zložitosť navrhnutých algoritmov.

1.1 Jednoduché grafy

Na začiatku si zadefinujeme niekoľko dôležitých pojmov a definícií z teórie grafov, ktoré budeme v tejto práci používať. K definícií signovaných grafov sa dostaneme v ďalšej časti, najskôr si potrebujeme zadefinovať pojem jednoduchý graf. Väčšina pojmov v tejto kapitole je prebraných z knihy [1].

Jednoduchým grafom G nazveme usporiadanú dvojicu $G = (V, E)$, kde V je konečná množina vrcholov a E je neusporiadaná množina hrán, t.j. dvojíc typu $\{u, v\}$ takých, že $u, v \in V$ a $u \neq v$.

Rád grafu G je počet jeho vrcholov, označuje sa $|G|$. Počet hrán grafu G označujeme $||G||$.



Obr. 1: *Tri jednoduché grafy*

Na obrázku 1 vidíme príklady troch jednoduchých grafov s očíslovanými vrcholmi. Napríklad graf $G = (V, E)$ na obrázku 1.c) má $V = \{0, 1, 2, 3, 4\}$ a $E = \{\{0, 1\}, \{1, 2\}, \{0, 2\}, \{3, 4\}\}$.

Vrchol u je *incidentný* s hranou e, ak $u \in e$. Dva vrcholy u a v sú susedné, ak $\{u, v\} \in E(G)$. Dve hrany $e \neq f$ sú susedné, ak majú spoločný vrchol, t.j. $\exists v \in V(G)$ taký, že $v \in e$ a $v \in f$. Všetky vrcholy, s ktorými daný vrchol u susedí, sú jeho susedia alebo okolie $O(u) = \{v : \{u, v\} \in E\}$.

Stupeň vrchola $u \in V$ je rovný počtu hrán s ním incidentných. Stupeň vrchola u sa označuje *deg(u)*.

Graf G sa nazýva *kompletný*, ak všetky jeho vrcholy sú navzájom susedné. Kompletný graf s n vrcholmi označujeme K_n. Na obrázku 1.b) vidíme kompletný graf K_5 s 5 vrcholmi.

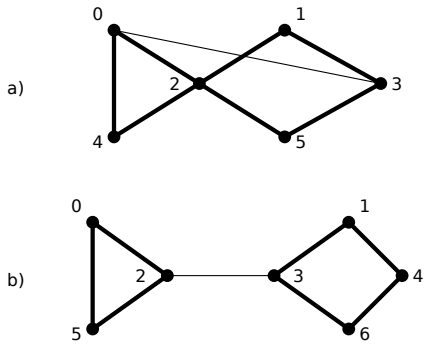
Sled S v G nazveme striedavú postupnosť vrcholov a hrán $v_0, e_1, v_1, e_2, v_2, \ldots, v_{p-1}, e_p, v_p$ takú, že $e_i = \{v_{i-1}, v_i\}$. Hovoríme, že sled začína vo v_0 a končí vo v_p. Ak $v_0 = v_p$ sled nazývame uzavretý. Sled S sa nazýva *ťah*, ak hrany sledu sú rôzne. Sled S sa nazýva *cesta*, ak vrcholy sledu sú rôzne. Číslo p sa nazýva *dĺžka sledu*. Cestu niekedy označujeme aj ako postupnosť vrcholov $v_0\ v_1\ \ldots\ v_p$.

Graf G je *súvislý*, ak medzi ľubovoľnými dvoma vrcholmi $u, v \in V$ existuje sled z vrchola u do vrchola v. Na obrázku 1.a) a 1.b) sú príklady súvislých grafov.

Komponent G je súvislý podgraf grafu G, ktorý je maximálny vzhľadom na inklúziu.

Hovoríme, že graf je *nesúvislý*, ak nie je súvislý. Takýto graf je tvorený aspoň dvoma komponentami. Na obrázku 1.c) vidíme príklad nesúvislého grafu, ktorý je tvorený dvoma komponentami.

Majme graf G a $V(G) = \{x_0, x_1, x_2, \ldots, x_{n-1}, x_n\}$. Ak $P = x_i x_{i+1} \ldots x_{j-1} x_j$ je cesta a $(j - i + 1) \geq 3$, potom podgraf $C := P + x_j x_i$ nazývame *kružnica*. Dĺžka kružnice je počet jej hrán (alebo vrcholov). Kružnica dĺžky k sa nazýva k - *kružnica* a označuje sa C_k.



Obr. 2: *Kružnice zvýraznené hrubými čiarami.*

Na obrázku 2.a) je vidno kružnice: 0,2,4,0 a 2,1,3,5,2, naopak kružnicou nie je uzavretý sled 0,2,1,3,5,2,4,0, ani 0,2,5,3,1,2,4,0, lebo vrchol 2 sa v týchto dvoch postupnostiach opakuje dvakrát.

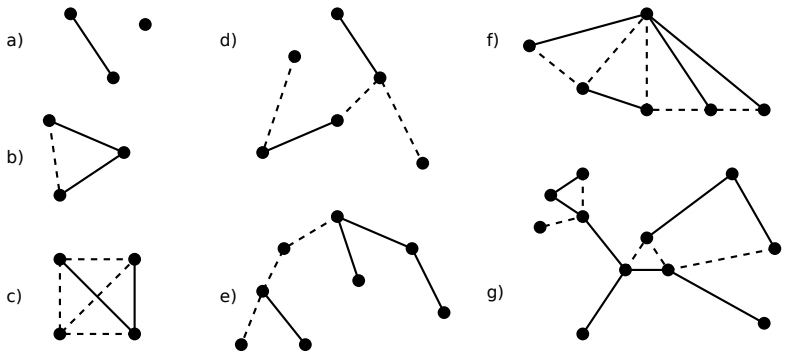
Graf na obrázku 2.b) obsahuje kružnice: 0,2,5,0 a 3,1,4,6,3, naopak kružnicou nie je napríklad postupnosť vrcholov 0,2,3,1,4,6,3,2,5,0, lebo hrana $\{2, 3\}$ sa v tejto postupnosti opakuje.

1.2 Signované grafy

V predchádzajúcej časti sme definovali graf ako množinu vrcholov, z ktorých niektoré sú spojené hranou. Vrcholy zvyčajne reprezentujú objekty alebo entity zatiaľ čo hrany určujú vzťah medzi vrcholmi. Napríklad, ak opisujeme medziľudské vzťahy, vrcholy môžu reprezentovať jedincov a hrany vzťahy medzi jedincami.

V tejto kapitole definujeme nový objekt signovaný graf, ako jednoduchý graf, kde každá hrana je buď kladná alebo záporná. Signované grafy sú vhodné napríklad na reprezentáciu symetrickej binárnej relácie medzi entitami, respektíve, ak je vzťah medzi každými dvomi vrcholmi symetrický. So signovanými grafmi môžeme modelovať napr. osobné vzťahy medzi ľuďmi. Konkrétne, signované grafy sú vhodné na opis opačných medziľudských vzťahov, ako sú mať rád - nemať rád, priateľstvo - neznášanlivosť atď.

Formálne je *signovaný graf* G trojica (V, E, Σ), kde V je množina vrcholov, E je množina hrán a zobrazenie Σ, definované predpisom $\Sigma : E(G) \to \{+1, -1\}$ priradí každej hrane grafu G kladné respektíve záporné znamienko [4].
Inak povedané, pre daný graf $H = (V(H), E(H))$, vieme zadefinovať signovaný graf G s podkladovým grafom H ako $G = (V(H), E(H), \Sigma)$, alebo skrátene $G = (H, \Sigma)$.



Obr. 3: *Signované grafy nakreslené so zápornými hranami prerušovanou čiarou*

Často sa hrany signovaných grafov kreslia dvomi farbami, kde jedna farba je pre kladné hrany, druhá pre záporné. V tejto práci budú záporné hrany nakreslené prerušovanými čiarami a kladné hrany plnými čiarami.

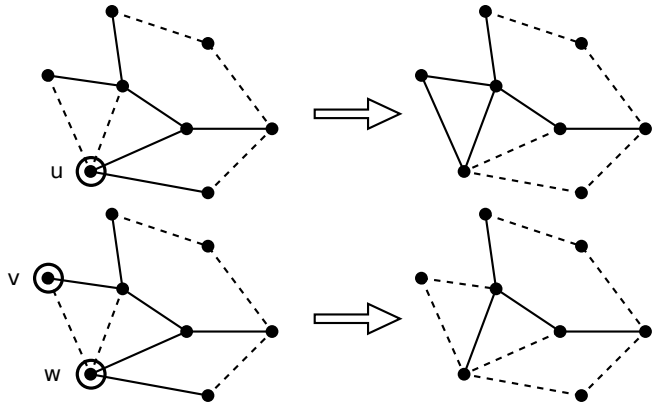
Hovoríme, že kružnica C v signovanom grafe G je *balansovaná*, ak obsahuje párny počet záporných hrán a *nebalansovaná*, ak obsahuje nepárny počet záporných hrán. Signovaný graf, alebo podgraf je *vyvážený*, ak každá kružnica je balansovaná. Základnou charakteristikou signovaných grafov je množina balansovaných kružníc, označená ako $B(G)$. Dve základné otázky týkajúce sa signovaných grafov znejú: Je signovaný graf vyvážený? Aký je najväčší podgraf signovaného grafu, ktorý je vyvážený? Odpoveď na prvú otázku nie je zložitá, druhá je výpočtovo neriešiteľná, pretože je to *NP-ťažká* úloha [7].

Signované grafy boli poprvykrát predstavevé Hararym v roku 1956, ktorý ich použil na reprezentáciu problému v sociálnej psychológii (Cartwright, Harary [8]). Signované grafy boli znovu objavené mnohokrát, pretože sa prirodzene objavovali v nesúvisiacich oblastiach. Napríklad umožňujú popísať a analyzovať geometriu podmnožín koreňových systémov. Balansované a nebalansované kružnice v signovaných grafoch predstavujú prirodzený ekvivalent k otázke ohľadom párnych a nepárnych kružníc v grafoch. Objavujú sa aj v topologickej teórií grafov a teórií grúp, pri výpočte základného stavu energie v neferomagnetickom Isingovom modeli - kde je treba nájsť najväčšiu vyváženú množinu hrán G. Signované grafy boli taktiež použité ku klasifikácií dát v korelácií klastrov.

1.2.1 Prepínanie vrcholov

Dôležitou operáciou na signovaných grafoch je prepínanie vrcholov. Nech u je vrchol signovaného grafu G. Prepnutie vrchola u je operácia, ktorá zneguje znamienka na jeho incidentných hranách. Teda na každej hrane incidentnej s vrcholom u zmení znamienko na opačné.

Definujme prepínanie vrcholov na množine vrcholov W signovaného grafu G tak, že každý vrchol z W prepneme práve raz. Je to ekvivalentné s tým, že postupne prepneme každý z vrcholov množiny W práve raz.



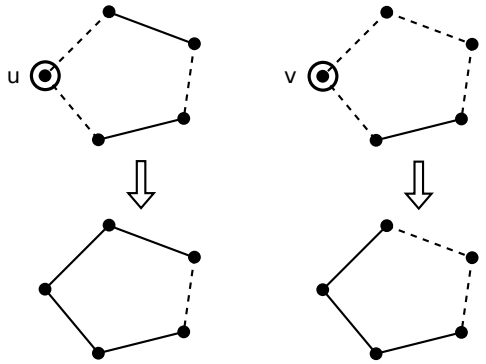
Obr. 4: *Prepnutie vrchola u, resp. prepnutie vrcholov v a w*

Na obrázku 4 hore vidíme prepnutie zakrúžkovaného vrchola u, dolný obrázok ukazuje prepnutie dvoch zakrúžkovaných vrcholov v a w. Možno si všimnúť, že po prepnutí susedných vrcholov v a w, zostala hrana $\{v, w\}$ nezmenená.

Tvrdenie: Prepnutím $k = |W|$ vrcholov na signovanom grafe G sa zmení Σ na Σ' rovnako ako prepnutím komplementu, t.j. $n - k$ vrcholov, $n = V(G),\ k \leq n$.
Dôkaz: Nech G je signovaný graf, $V(G) = n$ a u je jeden pevný vrchol signovaného grafu G a nech $deg(u) = l$. Prepnutím u sa znegujú znamienka na jeho l incidentných hranách, teda Σ sa zmení na Σ' tak, že Σ' priradí všetkým incidentným hranám vrchola u opačné znamienka ako Σ a pre ostatné hrany zostanú priradenia rovnaké ako v Σ. Chceme, aby aj po prepnutí vrcholov množiny $V(G) \setminus \{u\}$ sme dostali Σ'. Vieme, že ak prepneme dva susedné vrcholy, hrana ktorú tvoria zostane nezmenená. Preto ak prepneme všetky vrcholy množiny $V(G) \setminus \{u\}$, potom všetky hrany okrem l hrán incidentných s u zostanú nezmenené. Teda dostaneme presne Σ'.

Uvažujeme teraz prepnutie k vrcholov na G, pričom $k \in \{1, 2, \ldots, |W|\}$. Prepnutím k vrcholov sa Σ zmení na Σ' takto: hranám, ktoré nemajú koncové vrcholy v $|W|$, zostanú priradené rovnaké znamienka ako v Σ, hranám, ktoré majú jeden koniec v W budú mať v Σ' priradené opačné znamienka ako v Σ a hranám, ktoré majú obidva konce v W, budú mať v Σ' rovnaké znamienka ako v Σ. Teraz ak uvažujeme prepnutie $n-k$ vrcholov grafu G, potom sa Σ zmení na Σ' takto: hrany, ktoré majú obidva konce v množine $V \setminus W$ budú mať v Σ' priradené rovnaké znamienka ako v Σ, hrany ktoré nemajú ani jeden koniec v $V \setminus W$ budú mať priradené v Σ' rovnaké znamienka ako v Σ a hrany, ktoré majú práve jeden koniec v $V \setminus W$ budú mať v Σ' opačné znamienka ako v Σ. Hrany, ktoré majú jeden koniec v $V \setminus W$ musia mať druhý koniec v W a tie, ktoré majú jeden koniec v W musia mať druhý koniec v $V \setminus W$, teda sú to tie isté hrany, ktorým sa v Σ' zmenia znamienka na opačné. □

Prepnutím vrchola u kružnice C v signovanom grafe G sa zmenia znamienka na dvoch hranách incidentných s vrcholom u kružnice C, avšak parita záporných hrán C zostane rovnaká. Balansovaná kružnica preto zostane balansovaná a nebalansovaná zostane nebalansovaná. Množina nebalansovaných kružníc je invariantom vzhľadom na prepnutie vo vrcholoch.



Obr. 5: *Prepínanie vrcholov na kružnici*

Pri ľubovoľnom prepnutí vrcholov na kružnici sa jej balansovanosť nemení. Na obrázku 5 hore vľavo je nebalansovaná kružnica a po prepnutí vrchola u zostane opäť nebalansovaná. Vpravo vidíme prepnutie vrchola v na balansovanej kružnici.

Kapitola 2

2 Reprezentácia grafov v počítači

Ak chceme pohodlne narábať s grafmi, je výhodné, aby boli nejakým konkrétnym spôsobom zakódované a uložené v nejakej dátovej štruktúre. Poznáme niekoľko spôsobov, ako možno udržiavať a následne pracovať s grafmi t.j. grafovými štruktúrami v počítačoch. Pretože graf je štruktúra vrcholov (bodov alebo uzlov) a hrán (vzťahov medzi nimi), je dobré zvoliť správnu reprezentáciu takejto štruktúry.

2.1 Maticová reprezentácia

Jedným zo spôsobov, ako vhodne reprezentovať jednoduchý graf v počítači je pomocou matíc. Maticu A máme ako dvojrozmerné pole typu $m \times n$, kde každá položka vektora dĺžky m je vektor dĺžky n. Každý prvok matice A označíme ako $a_{i,j}$, kde $i \in \{1, 2, \ldots, m\}$ a $j \in \{1, 2, \ldots, n\}$. Pre maticovú reprezentáciu grafu máme niekoľko typov matíc.

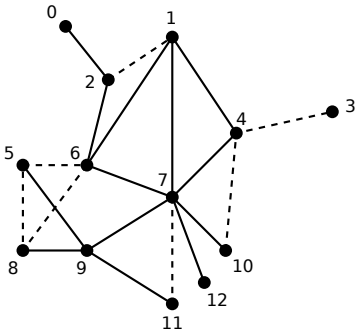
2.1.1 Matica susednosti

Alebo *Matrix Adjacency* je štvorcová matica typu $n \times n$, kde n je počet vrcholov grafu G a $v_1, \ldots, v_n$ sú jeho vrcholy [3]. Označuje sa $A(G) = (a_{i,j})$ a pre prvok $a_{i,j}$ matice A platí : $a_{i,j} = 1$ práve vtedy, keď vrcholy v_i a v_j sú susedné v grafe G. V opačnom prípade $a_{i,j} = 0$. Ak uvažujeme graf bez slučiek, tak A má na diagonále nuly. Čím má graf menej hrán, tým má táto matica viac núl a teda má väčšiu pamäťovú náročnosť oproti iným typom dátových štruktúr.

2.1.2 Matica susednosti pre signovaný graf

Maticu susednosti A^s pre signovaný graf je vhodné vyplniť symbolmi 0/+/- kde +/- použijeme pre kladnú/zápornú hranu medzi v_i a v_j a 0 podobne ako v matici susednosti pre jednoduché grafy. Ak by sme mali každý prvok matice A typu *boolean* pre 0 a 1, potom by každý prvok matice A zaberal v pamäti 1 byte . Keďže symboly +/- sú typu

char, potom každý prvok matice A^s je typu *char* namiesto *boolean* u jednoduchých grafov. Je zrejmé, že matica A^s bude mať väčšiu pamäťovú náročnosť ako matica A, pretože jeden jej prvok $a^s_{i,j}$ typu *char* zaberá 2 byte v pamäti oproti jednému prvku $a_{i,j}$ typu *boolean*, ktorý zaberá 1 byte v pamäti.



	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	+	0	0	0	0	0	0	0	0	0	0
1	0	0	-	0	+	0	+	+	0	0	0	0	0
2	+	-	0	0	0	0	+	0	0	0	0	0	0
3	0	0	0	0	-	0	0	0	0	0	0	0	0
4	0	+	0	-	0	0	0	+	0	0	-	0	0
5	0	0	0	0	0	0	-	0	-	+	0	0	0
6	0	+	+	0	0	-	0	+	-	0	0	0	0
7	0	+	0	0	+	0	+	0	0	+	+	-	+
8	0	0	0	0	0	-	-	0	0	+	0	0	0
9	0	0	0	0	0	+	0	+	+	0	0	+	0
10	0	0	0	0	-	0	0	+	0	0	0	0	0
11	0	0	0	0	0	0	0	-	0	+	0	0	0
12	0	0	0	0	0	0	0	+	0	0	0	0	0

Obr. 6: *Signovaný graf a jeho matica susednosti*

Na obrázku 6 máme signovaný graf a maticu susednosti, v ktorej je tento signovaný graf zakódovaný. Kladná hrana $\{i, j\}$ má v i-tom riadku a j-tom stĺpci, respektíve v j-tom riadku a i-tom stĺpci symbol $+$. Záporná hrana $\{k, l\}$ má v k-tom riadku a l-tom stĺpci, resp. v l-tom riadku a k-tom stĺpci symbol -. Je vidno, že táto matica je podľa diagonály symetrická a navyše obsahuje veľa núl. Z tohoto hľadiska je takéto kódovanie signovaných grafov pamäťovo náročné.

2.1.3 Matica vzdialenosti

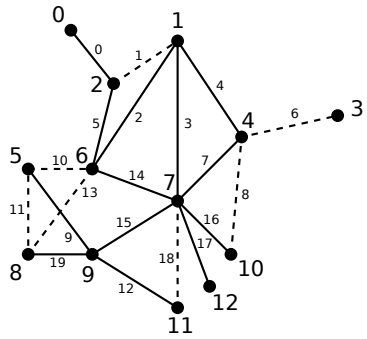
Obdobou matice susednosti je *matica vzdialenosti* A^d. Prvky matice $A^d = a^d_{i,j}$ v tomto prípade obsahujú ohodnotenia jednotlivých hrán (napr. kladné celé čísla) resp. príznak, že príslušná hrana v grafe neexistuje (napr. 0 alebo iná dostatočne malá alebo dostatočne veľká konštanta). Ak vrcholy u_i a u_j tvoria v grafe hranu, potom $a^d_{i,j} = x$, kde x je valuácia hrany, teda ohodnotenie rôzne od 0. Ak $a^d_{i,j} = 0$, potom u_i a u_j netvoria hranu.

Ak by mali signované grafy na hranách okrem znamienok aj iné atribúty, potom by sa dali maticou vzdialenosti zadať tak, že by sme maticu A^d s prvkami $a^d_{i,j}$ rozšírili na 3-rozmerné pole s prvkami $a^d_{i,j,k}$, kde prvé dve dimenzie i, j by určovali hranu a tretia dimenzia k by bolo pole dĺžky l s l atribútmi. Ak by signované grafy mali na hranách iba 2 atribúty - znamienka a nezáporné ohodnotenia, potom by sme mohli obidva atribúty zakódovať do jedného a vystačili by sme si s dvojrozmerným poľom.

Ak by hrana $\{u_i, u_j\}$ bola záporná a mala by valuáciu 5, potom by v matici A^d mal prvok $a^d_{i,j}$ hodnotu -5.

2.1.4 Matica priľahlosti

Známa ako *Matrix Incidency* je matica typu $n \times m$, ktorá sa označuje $A'(G)$ s prvkami $(a'_{i,j})$, kde n je počet vrcholov a m je počet hrán. Riadky matice A' zodpovedajú vrcholom grafu G a stĺpce matice A' zodpovedajú hranám. Pre prvok matice A' platí: $a'_{i,j} = 1$ práve vtedy, keď vrchol u_i je incidentný s hranou e_j v grafe G. V opačnom prípade je $a'_{i,j} = 0$. Ak hrana existuje, potom je incidentná s práve dvomi vrcholmi, ktoré sa nachádzajú medzi n riadkami tejto matice. Z toho vyplýva, že táto matica obsahuje počet príznakov rôznych od nuly rovný dva krát počtu hrán. Pre neorientované grafy sú všetky príznaky napr. 1 a orientované grafy môžu byť pomocou tejto matice reprezentované +1 a -1 podľa toho z ktorého vrcholu do ktorého vrcholu je hrana orientovaná. Špeciálne pre signované grafy môžu byť ako príznaky iba znamienka +/- podľa toho aké má hrana znamienko. Je zrejmé, že pre kladnú hranu $e_k = \{u_i, u_j\}$ je $a'_{i,k} = a'_{j,k} = +$.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	-	+	+	+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	+	-	0	0	0	+	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	-	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	+	0	-	+	-	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	+	-	-	0	0	0	0	0	0	0	0
6	0	0	+	0	0	+	0	0	0	0	-	0	0	-	+	0	0	0	0	0
7	0	0	0	+	0	0	0	+	0	0	0	0	0	0	+	+	+	+	-	0
8	0	0	0	0	0	0	0	0	0	0	0	-	0	-	0	0	0	0	0	+
9	0	0	0	0	0	0	0	0	0	+	0	0	+	0	0	+	0	0	0	+
10	0	0	0	0	0	0	0	0	-	0	0	0	0	0	0	0	+	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	+	0	0	0	0	0	-	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	+	0	0

Obr. 7: *Signovaný graf a jeho matica priľahlosti*

Na obrázku 7 máme rovnaký signovaný graf ako na obrázku 6, ale zakódovaný v matici priľahlosti. Veľkými číslami sú označené vrcholy a malými číslami hrany. Je vidno, že táto matica je väčšia, pretože má viac stĺpcov a rovnaký počet riadkov v porovnaní s maticou susednosti. Počet nenulových políčok je malý v porovnaní s nulovými políčkami. Táto matica uľahčuje určenie vrcholov, ktoré sú incidentné s hranou.

2.2 Zoznamy susedov, zoznamy hrán

Vďaka priamemu prístupu k prvkom matice vieme napr. ľahko zistiť prítomnosť hrany v grafe prostredníctvom riadkových a stĺpcových indexov. Takisto sa matice ľahko pro-

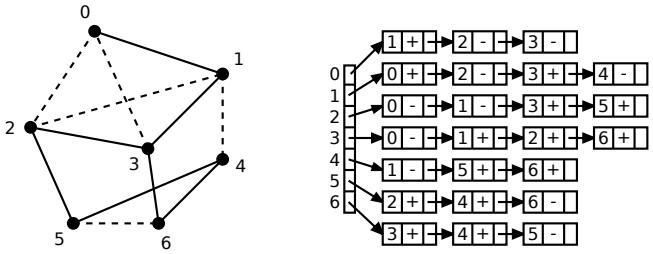
gramujú ako dvojrozmerné pole, kde každý prvok je rovnakého dátového typu. Pri grafoch s veľkým počtom vrcholov a relatívne malým počtom hrán môže maticová reprezentácia znamenať zbytočné plytvanie pamäte. Grafy s malým počtom hrán možno oveľa úspornejšie reprezentovať pomocou zoznamov susedov a zoznamov hrán.

V takejto reprezentácií grafu si pre každý z n vrcholov pamätáme v jednosmernom lineárnom zozname všetky jeho susedné vrcholy. Ak máme jednorozmerné pole s n prvkami, potom indexy určujú vrcholy grafu a každý prvok tohoto poľa obsahuje pointer na zoznam susedov vrchola určeného daným indexom.

Pre jeden vrchol i signovaného grafu G máme jednosmerný lineárny zoznam vrcholov s ním incidentných. V takomto zozname pre každý vrchol incidentný s vrcholom i máme záznam obsahujúci *id* vrchola, znamienko a pointer na další záznam (ďaľšieho suseda vrchola i). Posledný záznam v tomto jednosmernom lineárnom zozname obsahuje pointer nastavený na null[1], t.j. tu sa prehľadávanie susedov vrchola i končí.

Ak chceme zistiť, či sa v grafe G nachádza hrana $\{i, j\}$, potom stačí prejsť i-ty zoznam susedov a skontrolovať, či sa v niektorej položke prehľadávaného zoznamu nachádza vrchol j. Ak po prejdení na posledný záznam prehľadávaného i-teho zoznamu nenájde vrchol j, potom sa hrana $\{i, j\}$ v grafe G nenachádza.

Podobne by sa dalo zistiť prítomnosť hrany $\{i, j\}$ v grafe G aj pri prehľadávaní j-teho zoznamu, pretože každá hrana $\{i, j\}$ je evidovaná dvakrát (v i-tom aj j-tom zozname susedov). Táto operácia vyžaduje čas $\max(O(deg(i)), O(deg(j))$, kde $deg(i)$ je stupeň i - teho vrchola a $deg(j)$ je stupeň j-teho vrchola.



Obr. 8: *Signovaný graf a jeho zoznam susedov*

[1]Hovoríme, že pointer je nastavený na null, ak neobsahuje žiadnu adresu v pamäti (nikam neukazuje).

Vytvorenie takejto štruktúry signovaného grafu G nie je ťažké na implementáciu, ale dá sa aj zjednodušiť vynechaním práce s pointermi, ak poznáme $\Delta(G)$ - maximálny stupeň signovaného grafu G. Mali by sme maticu typu $n \times \Delta(G)$ a o každom vrchole zapamätaný jeho stupeň. Susedia vrchola u by boli uložení od začiatku v u - tom riadku a prehľadávanie zoznamu susedov vrchola u by sa realizovalo jednoduchým zvyšovaním indexu v stĺpci od 1 po $deg(u)$.

Ak uvažujeme ako ešte jednoduchšie a úspornejšie uchovať signovaný graf G v počítači, môžeme použiť zoznam hrán uložený v jednoduchom poli. Toto jednorozmerné lineárne pole dĺžky $3 \cdot |E(G)|$ má každú hranu určenú ako trojicu susedných prvkov v určujúcom poradí, kde prvé dva prvky sú id vrcholov incidentných s touto hranou a tretí prvok je znamienko prislúchajúce tejto hrane. Prvá hrana zaberá v tomto poli 1.,3. položku, druhá hrana 4.,6. položku atď. Takto zakódovaný signovaný graf G zaberá minimum miesta v pamäti, ale za cenu väčšej námahy s vykonávaním príslušných operácií na grafoch. Ak chceme napríklad zistiť $deg(u)$ pre nejaký vrchol u, treba prejsť celé pole a sčítať počet výskytov vrchola u. Pritom v zozname susedov by sme v zozname vrchola u prešli na posledný záznam a vedeli by sme koľko záznamov zoznamu susedov vrchola u sme navštívili a teda aký je $deg(u)$.

Zoznam hrán pre signované grafy sa dá s rovnakou pamäťovou náročnosťou zakódovať do dvojrozmerného poľa $|E(G)| \times 3$. Tu máme ľahké nájsť hranu podľa id hrany, kde $id \in \{0, \ldots, |E(G)| - 1\}$ sú indexy prvej dimenzie. Ak chceme zistiť, či sú dva vrcholy u_1 a u_2 incidentné, stačilo by v niektorej z $|E(G)|$ trojíc nájsť u_1 a súčasne u_2.

2.3 Dátový typ *igraph_t*

Dátový typ *igraph_t* sa nachádza v knižnici *iGraph*, ktorej sa budeme podrobnejšie venovať v nasledujúcich kapitolách, kde ju budeme potrebovať pre nájdenie izomorfizmu jednoduchých grafov [16]. Dátový typ *igraph_t* je interná, jednoduchá a účinná dátová štruktúra pre ukladanie grafov. Ak máme v programe graf ako inštanciu typu *igraph_t*, nemusíme robiť žiadne matice ani zoznamy a zároveň máme k dispozícií postačujúce množstvo funkcií z knižnice *iGraph* na rôzne operácie na grafoch. Dátový typ *igraph_t* je deklarovaný v hlavičkovom súbore *igraph_datatype.h* nachádzajúcom sa v adresári *include* knižnice *iGraph* nasledovne :

```
typedef struct igraph_s {
  igraph_integer_t n; /*počet vrcholov*/
  igraph_bool_t directed; /*či je graf orientovaný*/
  igraph_vector_t from; /*prvý stĺpec zoznamu hrán*/
  igraph_vector_t to; /*druhý stĺpec zoznamu hrán*/
  igraph_vector_t oi;
  igraph_vector_t ii;
  igraph_vector_t os;
  igraph_vector_t is;
  void *attr; /*pointer pre d'alšie rozšírenie (napr. znamienka)*/
{ igraph_t;
```

Vektory *oi, ii, os, is* majú tieto významy:
oi - index zoznamu hrán podľa prvého stĺpca, teda prvá hrana ($id = 0$) podľa tohoto poradia ide z *from[oi[0]]* do *to[oi[0]]*,
ii - index zoznamu hrán podľa druhého stĺpca, dĺžka tohoto vektora je rovnaká ako dĺžka vektora *oi*, rovná počtu hrán.
os - obsahuje pointery do zoznamu hrán *from* a *to* pre každý vrchol v. Prvá hrana vrchola v je hrana číslo *from[oi[os[v]]]* ak *os[v]<os[v+1]*. Ak *os[v]=os[v+1]*, potom z vrchola v nevychádzajú žiadne hrany. Dĺžka vektora *os* je rovná počtu vrcholov + 1, posledný element je počet hrán slúžiaci iba pre jednoduchšiu orientáciu,
is - analogicky ako *os*, ale pre vstupné hrany.

Kapitola 3

3 Problém izomorfizmu grafov

V tejto kapitole sa budeme zaoberať rozdielom medzi izomorfizmom jednoduchých grafov a izomorfizmom signovaných grafov. Potom si predstavíme niektoré známe algoritmy na nájdenie izomorfizmu jednoduchých grafov a stručne uvedieme ich hlavné myšlienky.

Problém izomorfizmu jednoduchých grafov je navrhnúť prakticky realizovateľný všeobecný algoritmus, ktorý by v rozumnom čase pre ľubovoľné dva grafy rozhodol, či sú izomorfné alebo nie, respektíve dokázať, že žiaden taký algoritmus neexistuje. Tento problém, označovaný ako *GI* problém je „ťažký“ napriek nejdnoduchosti definície. Patrí do triedy NP[2], ale nebolo dokázané, že patrí aj do triedy NP - úplnych problémov. Ak by sa ukázalo, že *GI* problém partí do triedy NP - úplnych problémov, viedlo by to k zrúteniu tejto klasifikácie a veľkým problémom súčasnej kryptografie (pozri [15]).

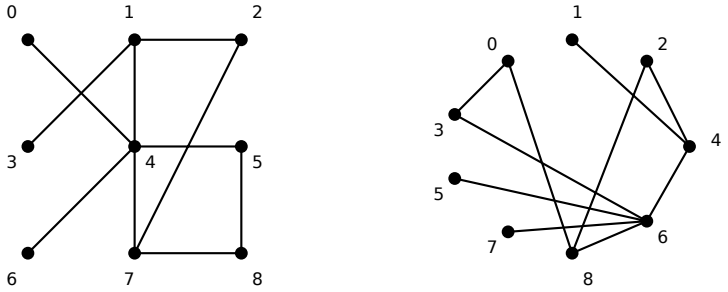
Problém izomorfizmu signovaných grafov je podobný ako pre jednoduché grafy, navyše máme znamienka na hranách, balansovanosť kružníc pre cyklické podkladové grafy a možnosť prepínania vrcholov.

3.1 Izomorfizmus jednoduchých grafov

Jednoduché grafy $G_1 = (V_1, E_1)$ a $G_2 = (V_2, E_2)$ nazývame izomorfné (označujeme $G_1 \cong G_2$), ak existuje bijekcia[3] $\varphi : V_1 \to V_2$, kde $\{u, v\} \in E_1 \Leftrightarrow \{\varphi(u), \varphi(v)\} \in E_2$ pre každé $u, v \in V_1$ [6].

[2]NP - (nedeterministicky polynomiálny) je trieda problémov, ktoré sa dajú riešiť v polynomiálnom obmedzenom čase na nedeterministickom počítači. Podobne možno aj v polynomiálnom čase overiť správnosť, obecne sa ale nedá nájsť riešenie v polynomiálnom čase pomocou klasického Turingovho stroja.

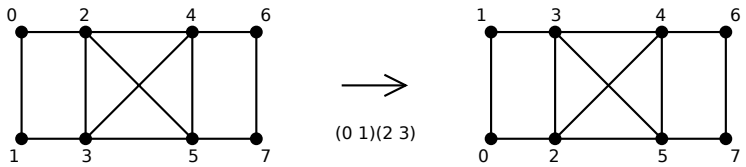
[3]Bijektívne zobrazenie priraďuje každému prvku z východiskovej množiny práve jeden prvok z cieľovej množiny a na každý prvok cieľovej množiny sa zobrazuje jeden prvok východiskovej množiny.



Obr. 9: *Dva izomorfné grafy*

Na obrázku 9 sú dva izomorfné grafy a φ_1 je izomorfizmus, kde : $\varphi_1(0) = 5$, $\varphi_1(1) = 4$, $\varphi_1(2) = 2$, $\varphi_1(3) = 1$, $\varphi_1(4) = 6$, $\varphi_1(5) = 3$, $\varphi_1(6) = 7$, $\varphi_1(7) = 8$, $\varphi_1(8) = 0$.
Existuje aj iný izomorfizmus, napríklad φ_2, kde : $\varphi_2(0) = 7$, $\varphi_2(1) = 4$, $\varphi_2(2) = 2$, $\varphi_2(3) = 1$, $\varphi_2(4) = 6$, $\varphi_2(5) = 3$, $\varphi_2(6) = 5$, $\varphi_2(7) = 8$, $\varphi_2(8) = 0$.

Automorfizmus grafu G je izomorfizmus G na seba samého. Inými slovami je to permutácia označení vrcholov, pričom je zachovaná množina hrán. Množina všetkých automorfizmov grafu G určuje *grupu*[4], kde grupová operácia je kompozícia permutácií. Táto grupa permutácií sa nazýva *grupa automorfizmov* grafu G a označuje sa $Aut(G)$. Množina generátorov pre $Aut(G)$ je najmenšia množina $S \subseteq Aut(G)$, kde každá permutácia okrem identity v $Aut(G)$ sa získa kompozíciou elementov z S. Výsledkom aplikácií dvoch automorfizmov je tiež nejaký automorfizmus [19].



Obr. 10: *Zámena označení vrcholov pri zachovaní hrán*

Na obrázku 10 sme zamenili označenie 0, 1 a 2, 3. Množina hrán zostala zachovaná, pretože po zámene napríklad zostal vrchol 2 susedný s vrcholom 5 a vrchol 0, ktorý nebol susedný s vrcholom 4 nie je s ním susedný ani po zámene. To znamená, že (0

[4]Grupa je množina G s binárnou asociatívnou operáciou $\circ$ a neutrálnym prvkom n taká, že pre ľubovoľný prvok $g \in G$ platí $g \circ n = n \circ g = g$ a ku každému prvku $g \in G$ existuje jeho inverzný prvok $g^{-1} \in G$ taký, že $g \circ g^{-1} = g^{-1} \circ g = n$. Grupa sa zvyčajne označuje $(G, \circ)$.

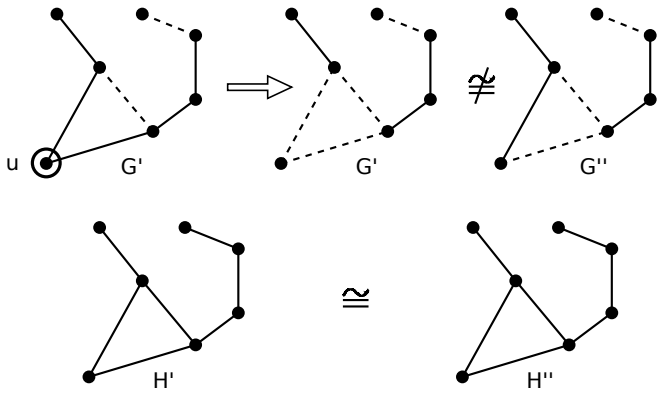
1)(2 3)[5] je automorfizmus. Grupa automorfizmov v grafe ne obrázku 10 obsahuje 8 automorfizmov: (1),(0 1)(2 3),(4 5)(6 7),(0 1)(2 3)(4 5)(6 7),(0 6)(1 7)(2 4)(3 5),(0 7)(2 5)(1 6)(3 4),(0 6 1 7)(2 4 3 5),(0 7 1 6)(2 5 3 4).

Pretože počet automorfizmov môže byť veľmi vysoký, je lepšie pracovať s generátormi *Aut(G)*. Pre graf na obrázku 10 množina generátorov *Aut(G)* je napríklad $S = \{(45)(67), (06)(17)(24)(35)\}$.

3.2 Izomorfizmus signovaných grafov

Hovoríme, že dva signované grafy $G_1 = (H_1, \Sigma_1)$ a $G_2 = (H_2, \Sigma_2)$ sú *izomorfné*, ak existuje izomorfizmus podkladového grafu $H_1 = (V_1, E_1)$ na podkladový graf $H_2 = (V_2, E_2)$, taký, že obraz každej kružnice C nachádzajúcej sa v G_1 je balansovaná kružnica C' v G_2 vtedy a len vtedy, ak C je balansovaná [5].

Signované grafy, ktoré majú podkladové grafy izomorfné nemusia byť izomorfné v signovanom zmysle. Hľadanie izomorfizmu signovaných grafov je to isté, ako keby sme hľadali vhodné prepnutie signovaných grafov a potom izomorfizmus podkladových grafov, ktorý zobrazí kladnú hranu na kladnú hranu a zápornú hranu na zápornú hranu. Platí to vďaka tomu, že prepnutie zachováva množinu balansovaných kružníc [11].

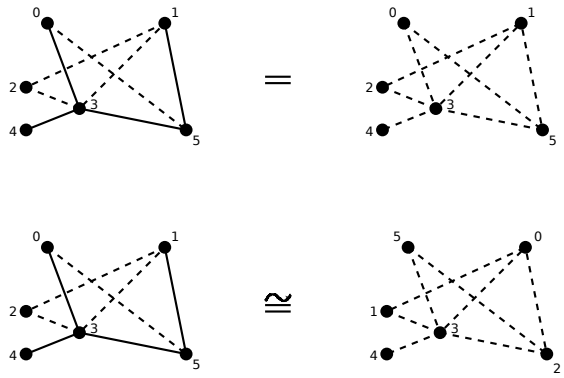


Obr. 11: *Izomorfizmus signovaných a jednoduchých grafov*

Na obrázku 11 máme signovaný graf G', ktorý sa nedá prepnúť na signovaný graf G'', pretože jediná kružnica v grafe G' je nebalansovaná a v grafe G'' balansovaná. Teda signovaný graf G' nie je izomorfný so signovaným grafom G''. Pre ich podkladové grafy H' a H'' je izomorfizmus zrejmý.

[5] Ak uvažujeme permutácie množiny $\{0, 1, 2, 3, 4, 5, 6, 7\}$, potom automorfizmus $\varphi = (0\ 1)(2\ 3)$ označuje permutáciu s vlastnosťou $\varphi(0) = 1$, $\varphi(1) = 0$, $\varphi(2) = 3$, $\varphi(3) = 2$, ktorá prvky 4, 5, 6, 7 nemení (teda $\varphi(4) = 4$, $\varphi(5) = 5$, $\varphi(6) = 6$, $\varphi(7) = 7$).

Hovoríme, že dva signované grafy $G_1 = (H, \Sigma_1)$ a $G_2 = (H, \Sigma_2)$ sú *ekvivalentné*, ak existujú vrcholy, ktorých prepnutím sa G_1 zmení na G_2. Inak povedané, dva signované grafy sú ekvivalentné, ak po prepnutí nejakej podmnožiny vrcholov dostaneme tie isté signované grafy. Na ekvivalenciu sa dá pozerať ako na špeciálny prípad izomorfizmu, kde zobrazenie, ktoré uvažujeme medzi vrcholovými množinami je identita. Ak nie sú signované grafy ekvivalentné, môžu byť ešte izomorfné.



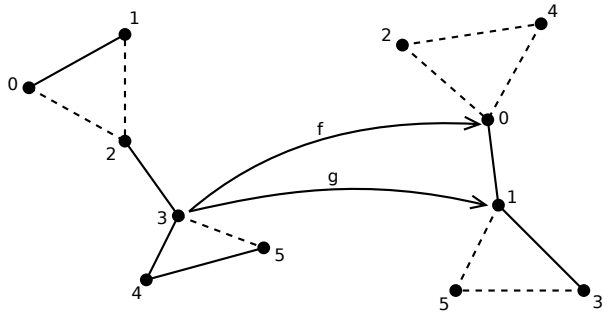
Obr. 12: *Ekvivalencia a izomorfizmus signovaných grafov*

Na obrázku 12 hore sú ekvivalentné signované grafy, pretože ich podkladové grafy sú rovnaké a ak prepneme vrcholy 1 2 a 3 na signovanom grafe hore vľavo, dostaneme celý záporný graf. Ak sú signované grafy ekvivalentné, sú zároveň aj izomorfné. Signované grafy na obrázku 12 dole sú izomorfné, ale nie ekvivalentné - vrchol s označením 1 naľavo má stupeň 3 a vrchol s rovnakým označením 1 napravo má stupeň 2. Ich podkladové grafy sú izomorfné a ak prepneme vrcholy 1 2 a 3 na signovanom grafe dole vľavo, zmení sa na celý záporný ako je nakreslený dole vpravo.

3.2.1 Bázové cykly a cyklový priestor

Pri definícií izomorfizmu signovaných grafov sme povedali, že pre izomorfizmus podkladových grafov musí aj každá kružnica na G_1 mať svoj obraz na G_2 s rovnakou paritou záporných hrán. Inak by neexistovalo prepnutie a signované grafy by neboli izomorfné.

Ak sa pre nejaký izomorfizmus φ podkladových grafov H_1 a H_2 zobrazí nejaká kružnica $C \in G_1$ na kružnicu $C' \in G_2$ s nerovnakou paritou záporných hrán, nezmená to, že G_1 a G_2 nie sú izomorfné. Pre iný izomorfizmus ψ podkladových grafov H_1 a H_2 sa totiž môže každá kružnica $C \in G_1$ zobrazovať na kružnicu $C' \in G_2$ s rovnakou paritou záporných hrán. Toto tvrdenie je ľahké spozorovať na obrázku 13.



Obr. 13: *Obrazy kružníc a ich balansovanosť*

Na obrázku 13 sú dva izomorfné signované grafy G_1 a G_2, pričom medzi ich podkladovými grafmi H_1 a H_2 existuje 8 izomorfizmov. Ak by sa pre niektorý izomorfizmus f podkladových grafov zobrazil vrchol 3 na vrchol 0, potom by sa nutne zobrazila kružnica $C = 3\ 4\ 5\ 3$ na kružnicu $C' = 0\ 2\ 4\ 0$. Kružnica C je nebalansovaná a jej obraz C' takisto nebalansovaná. Druhá kružnica grafu G_1, ktorá je balansovaná sa zobrazuje na balansovanú kružnicu grafu G_2. Teda pre izomorfizmus φ podkladových grafov H_1 a H_2, kde $\varphi(0) = 3, \varphi(1) = 5, \varphi(2) = 1, \varphi(3) = 0, \varphi(4) = 2, \varphi(5) = 4$ sú signované grafy G_1 a G_2 izomorfné. V tomto prípade, ak sa vrchol 3 zobrazuje na 0, takéto priradenie majú 4 izomorfizmy podkladových grafov a pre každý z nich sú aj signované grafy izomorfné.

Ďalšie 4 izomorfizmy podkladových grafov sú tie, kde sa vrchol 3 zobrazí na vrchol 1 (zobrazenie g). Vtedy sa nebalansovaná kružnica 3 4 5 3 zobrazí na balansovanú 1 5 3 1. Preto pre tieto 4 izomorfizmy podkladových grafov, v ktorých sa vrchol 3 zobrazí na vrchol 1, nie sú signované grafy izomorfné.

Tvrdenie: Ak sú dva signované grafy $G_1 = (H_1, \Sigma_1)$ a $G_2 = (H_2, \Sigma_2)$ acyklické a existuje medzi ich podkladovými grafmi H_1 a H_2 izomorfizmus φ, potom G_1 a G_2 sú izomorfné a G_1 sa dá prepnúť na G_2.

Dôkaz: Nech G_1 a G_2 sú dva súvislé acyklické signované grafy a φ je izomorfizmus ich podkladových grafov H_1 a H_2. Keďže G_1 a G_2 sú acyklické, G_1 má jedinú kostru T_1 a G_2 má jedinú kostru T_2, pričom $T_1 = (V(H_1), E(H_1))$ a $T_2 = (V(H_2), E(H_2))$. Chceme nájsť množinu vrcholov $W \subseteq V(H_1)$ takú, že prepnutím vrcholov W sa Σ_1 zmení na Σ_2. Vezmime a zafixujme ľubovoľný vrchol $u \in V(H_1)$ a nech u je koreň T_1. Z u vedie do každého listu v_i práve jedna cesta P_i. Pri prechádzke po cestách P_i z koreňa u do listu v_i algoritmus prejde a skontroluje všetky hrany G_1, niektoré možno viackrát. Ak by v ceste P_j objavil hranu e, ktorej obraz e' podľa φ by mal opačné znamienko, prepol

by vrchol hrany e, ktorý je vo väčšej vzdialenosti od koreňa u. Ak by mala cesta P_j všetky hrany opačné, postupne by prepínal vrcholy P_j, pokiaľ by sa nedostal k listu v_j. Takýmto spôsobom by algoritmus poprepínal všetky vrcholy množiny W a všetky znamienka na hranách G_1 by boli rovnaké ako ich obrazy na G_2, teda Σ_1 by bola identická so Σ_2. □

Nech T je kostra G, pričom G je cyklický signovaný graf. Každá hrana $e \in G$, ktorá nepatrí T je mimokostrová hrana. Po pridaní jednej mimokostrovej hrany e do T vznikne práve jedna kružnica C a hovoríme, že C je *fundamentálna kružnica* alebo tiež *bázový cyklus*. Počet bázových cyklov je rovný počtu mimokostrových hrán. Všetky bázové cykly $C_1, C_2, \ldots, C_k$ tvoria bázu *cyklového priestoru* $C(G)$. Cyklový priestor je množina všetkých kružníc grafu a každú kružnicu vieme dostať ako *symetrický rozdiel* niekoľkých fundamentálnych kružníc. Symetrický rozdiel kružníc $C_i, C_{i+1}, \ldots, C_j$ je kružnica C, ktorá vznikne symetrickou diferenciou kružníc $C_i, C_{i+1}, \ldots, C_j$, presnejšie ich množín hrán[6].

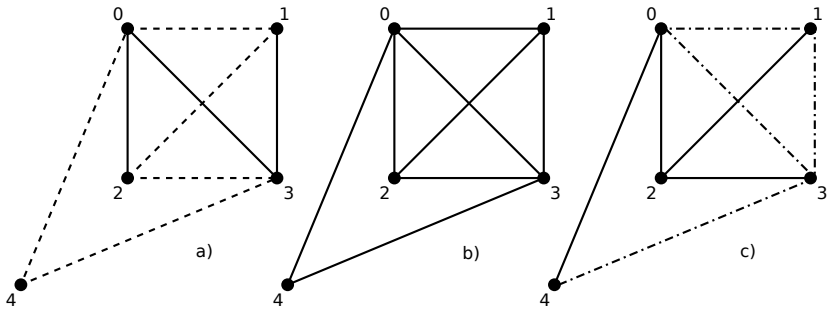
Tvrdenie: Parita záporných hrán bázových cyklov $C_1, C_2, \ldots, C_k$ z ktorých vznikne symetrickou diferenciou kružnica C je rovnaká ako parita záporných hrán C.
Dôkaz: Tvrdenie dokážeme matematickou indukciou vzhľadom na k.
Báza indukcie:
$(k = 1)$: kružnica C tvorená jediným bázovým cyklom C_1 má triviálne rovnaký počet záporných hrán a $C = C_1$.
$(k = 2)$: kružnica C je tvorená dvomi bázovými cyklami $C_1,\ C_2$. Keďže C tvoria dva bázové cykly, potom $C_1 \cap C_2 \neq \emptyset$ a majú spoločnú aspoň jednu hranu. Ak by C_1 bola balansovaná a C_2 nebalansovaná, potom nech by ich prienik mal ľubovoľnú paritu záporných hrán, C by bola nebalansovaná. Ak by boli $C_1,\ C_2$ obidve balansované alebo obidve nebalansované a ich prienik by mal ľubovoľnú paritu záporných hrán, C by bola balansovaná. Je to tým, že ak od dvoch čísel $a,\ b$ rôznej parity odrátame rovnaké číslo c (nech c má ľubovoľnú paritu), potom čísla $a,\ b$ budú mať opäť rôzne parity. Podobne ak sú $a,\ b$ rovnakej parity a odrátame od nich c, tak $a,\ b$ budú mať opäť rovnakú paritu.
Indukčný predpoklad:
Predpokladajme tvrdenie pre k bázových cyklov, z ktorých symetrickou diferenciou dostaneme C. Teda ak počet záporných hrán bázových cyklov je párny, potom je C balansovaná a ak je počet záporných hrán bázových cyklov nepárny, potom je C neba-

[6]Symetrický rozdiel (symetrická diferencia) n-množín $A_1, A_2, \ldots, A_n$ je množina M takých prvkov, že z každej množiny A_i obsahuje tie prvky, ktoré sa nachádzajú iba v A_i a v žiadnej inej množine A_j, $i \neq j$. Symetrickú diferenciu dvoch množín A_1,A_2 označujeme $A_1 \triangle A_2$.

lansovaná.

Indukčný krok:

Dokážeme tvrdenie pre $k+1$ bázových cyklov, pričom tvrdenie dokážeme pre dve množiny s aplikovaním indukčného predpokladu na k bázových cyklov. Nech teda C dostaneme ako symetrický rozdiel bázových cyklov $C_1, C_2, \ldots, C_k, C_{k+1}$. Podľa indukčného predpokladu symetrickou diferenciou bázových cyklov $C_1, C_2, \ldots, C_k$ dostaneme kružnicu C', ktorá bude balansovaná, ak počet záporných hrán C_i je párny respektive C' bude nebalansovaná, ak počet záporných hrán C_i je nepárny, pre $i \in \{1, 2, \ldots, k\}$. Potom C bude $C' \triangle C_{k+1}$ a bez ohľadu na to akú paritu záporných hrán má $C' \cap C_{k+1}$, kružnica C bude balansovaná, ak sú obidve C' aj C_{k+1} súčasne alebo balansované alebo nebalansované a C bude nebalansovaná naopak. □



Obr. 14: *Balansovanosť nebázovej kružnice*

Na obrázku 14.a) je signovaný graf G, 14.b) je podkladový graf H signovaného grafu G a na 14.c) je kostra signovaného grafu G zvýraznená plnými čiarami a mimokostrové hrany sú vyznačené bodkočiarkovanými čiarami. Vidíme, že kružnica $C = 0\ 1\ 3\ 4\ 0$ má nepárny počet záporných hrán a preto je nebalansovaná. Ak sa pozrieme na obrázok c), vidíme, že C tvoria 3 mimokostrové hrany $\{0,1\}, \{1,3\}, \{3,4\}$ a jedna kostrová hrana $\{0,4\}$, teda C je tvorená tromi bázovými cyklami 0 1 2 0, 1 2 3 1, 0 2 3 4 0. Každá z nich je nabalansovaná a výsledná kružnica C je takisto nebalansovaná.

Tvrdenie: Predpokladajme, že máme dva signované grafy G_1 a G_2, ktoré sú izomorfné (t.j. medzi ich podkladovými grafmi H_1 a H_2 existuje izomorfizmus φ a každá kružnica C má svoj obraz $\varphi(C)$ rovnakej balansovanosti). Potom pre akúkoľvek kostru G_1 a fundamentálnu kružnicu C, bude aj $\varphi(C)$ mať rovnakú paritu záporných hrán ako C.
Dôkaz: Nech T je ľubovoľná kostra G_1. Keďže H_1 a H_2 sú izomorfné, potom obraz $\varphi(T)$ je kostra G_2. Uvažujme mimokostrovú hranu $e \in G_1$. Vieme, že e určuje bázový cyklus

C v G_1 a obrazom C je bázový cyklus $\varphi(C)$ s mimokostrovou hranou $\varphi(e)$. Pretože vo φ sa balansovaná kružnica zobrazuje na balansovanú a nebalansovaná na nebalansovanú, tak $\varphi(C)$ má rovnakú paritu záporných hrán ako C. Keďže sme uvažovali ľubovoľnú kostru T, tvrdenie sme dokázali pre každú kostru signovaného grafu G_1. □

Opačné Tvrdenie: Nech φ je zobrazenie G_1 na G_2 také, že φ je izomorfizmus podkladových grafov H_1, H_2 a navyše existuje kostra T grafu G_1 a kostra T' grafu G_2, pričom φ zobrazí každú fundamentálnu kružnicu $C \in G_1$ na $C' \in G_2$ rovnakej balansovanosti, potom φ je izomorfizmus G_1 a G_2.
Dôkaz: Keďže φ je izomorfizmus H_1, H_2, stačí ukázať, že obrazom každej kružnice $C \in G_1$ je kružnica $C' \in G_2$ s rovnakou paritou záporných hrán. Každú kružnicu C vieme dostať ako symetrický rozdiel $\triangle$ niekoľkých fundamentálnych kružníc. Nech C je teda kružnica, ktorá vznikne ako symetrický rozdiel fundamentálnych kružníc $C_1, \ldots, C_k$, ktoré majú znamienko $Z_1, \ldots, Z_k$, pričom C má znamienko Z[7]. (Teda Z určuje, či je C balansovaná alebo nie.) Potom $C = C_1 \triangle \cdots \triangle C_k$ a $\varphi(C) = \varphi(C_1 \triangle \cdots \triangle C_k)$. Pretože φ je izomorfizmus grafov, potom $\varphi(C) = \varphi(C_1) \triangle \cdots \triangle \varphi(C_k)$, kde $\varphi(C_i)$ má rovnaké znamienko Z_i ako C_i, pre každé i a teda $\varphi(C)$ má rovnaké znamienko Z ako C. □

Po predchádzajúcich úvahách sme dospeli k návrhu algoritmu, ktorý rozhodne, či sú alebo nie sú signované grafy izomorfné. Algoritmus bude teda pracovať iba s izomorfizmom podkladových grafov a bázovými cyklami. Keďže jeden izomorfizmus podkladových grafov nestačí, je potrebné vedieť nájsť postupne všetky izomorfizmy. K tomu použijeme existujúci algoritmus, ktorý nám ich nájde.

3.3 Niektoré algoritmy na nájdenie izomorfizmu jednoduchých grafov

Jeden spôsob ako nájsť izomorfizmus jednoduchých grafov je permutovať označenie vrcholov na jednom z nich. Tento algoritmus na nájdenie izomorfizmu má časovú zložitosť $O(n!)$, pretože pre pevné označenie vrcholov na G_1 spermutuje (funkcia γ) označenia vrcholov na G_2 a kontroluje, či obraz každej hrany $\{u, v\} \in E(G_1)$ je hrana $\{\gamma(u), \gamma(v)\} \in E(G_2)$ pre $|V(G_1)| = |V(G_2)|$ a $|E(G_1)| = |E(G_2)|$. Ak by G_1 a G_2 boli izomorfné, v najhoršom prípade by sa mohlo stať, že by našiel prvý izomorfizmus až niekde medzi poslednými možnosťami zo všetkých $n!$ označení a ak by neboli izomorfné, musel by prejsť dokonca všetky z $n!$ možností. Takýto algoritmus pre väčší počet vrcholov by počítal príliš dlho a preto je nepoužiteľný. Lepší algoritmus, ktorý

[7] Znamienko kružnice hovorí, či je kružnica balansovaná + alebo nebalansovaná -.

nemožno nespomenúť je *Ullmamov* algoritmus [14]. Ullmanov algorimus používa procedúru pre obmedzenie prehľadávacieho stavového priestoru založenú na jednoduchom predpoklade. Ak sú v grafe G dva vrcholy spojené hranou, potom môžu byť tieto dva vrcholy namapované opäť na dva vrcholy spojené hranou.

Ako už bolo spomenuté, máme k dispozícií algoritmy s lepšou - exponenciálnou časovou zložitosťou ako napríklad *Bliss* implementovaný v knižnici *iGraph*.

3.3.1 Nauty

Program *Nauty* je považovaný za najlepší používaný algoritmus na nájdenie izomorfizmu jednoduchých grafov. Program *Nauty* je *open source* projekt napísaný vo veľmi portabilnej podmnožine jazyka C a moderné kompilátory jazyka C pre väčšinu typov počítačov by nemali mať nijaké problémy s prekladom [18].

Nauty je sada procedúr hľadajúcich grupu automorfizmov $Aut(G)$, *vrcholovo*[8] *zafarbeného* grafu G. Prvkami $Aut(G)$ sú všetky automorfizmy G vrátane triviálneho, ktorý necháva všetky označenia na mieste - identita. Pretože počet automorfizmov môže byť príliš veľký, algoritmus pracuje so sadou generátorov $S \subseteq Aut(G)$.

Druhou významnou funkciou *Nauty* je *kanonické označenie*. Je to operácia označenia vrcholov spôsobom, ktorý nezávisí na tom, ako boli vrcholy označené predtým. Grafy, ktoré sú izomorfné (rovnaké okrem označenia vrcholov) sa stanú identické (presne rovnaké) po kanonickom označení. Účelom kanonického označenia je otestovať izomorfizmus: dva izomorfné grafy sa stanú identické, ak sú kanonicky označené.

Vrcholy grafov sú v *Nauty* zafarbené a definícia automorfizmu pre zafarbenie hovorí, že každý vrchol $u \in G_1$ môže byť namapovaný iba na vrchol $v \in G_2$ rovnakej farby. Farby sa aplikujú v určitom poradí, t.j. prvá farba, druhá farba, atď. Pri kanonickom označení, označenie nového vrchola sa realizuje podľa poradia farieb. Vrcholy prvej farby sú označované skôr ako vrcholy druhej farby atď. Toto pravidlo znamená, že kanonické označenie môže byť použité pre určenie, či dva vrcholovo zafarbené grafy sú izomorfné cez izomorfizmus, ktorý mapuje každý vrchol prvého grafu na vrchol rovnakej farby druhého grafu.

Nauty vie pracovať s orientovanými grafmi a aj s grafmi so slučkami[9]. Autormi *Nauty* sú *Brendan D. McKay* a *Adolfo Piperno*. Ich posledná verzia programu *Nauty* je v súčasnosti 2.5.

[8] Vrcholovo zafarbený graf má zafarbené vrcholy tak, že žiadne dva susedné vrcholy nemajú rovnakú farbu.

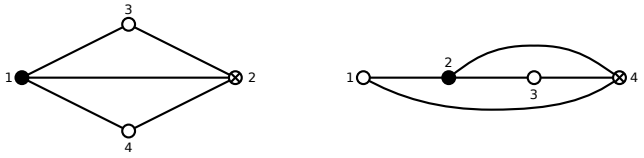
[9] Slučka je hrana e taká, že jej koncové vrcholy sú jeden a ten istý vrchol.

3.3.2 Bliss

Bliss je považovaný za nástupcu *Nauty* s lepšími heuristikami a dátovými štruktúrami. Je to *open source* projekt napísaný v jazyku C++. Počíta kanonické označenie grafov a grupu automorfizmov $Aut(G)$ vrcholovo zafarbeného grafu $G = (V, E, c)$, kde $c : V \to N$ je funkcia, ktorá priraďuje každému vrcholu nezáporné celé číslo (farbu). Automorfizmus vrcholovo zafarbeného grafu je izomorfizmus vrcholovo zafarbeného grafu samého na seba [20].

Autori vo svojej dokumentácií uvádzajú myšlienku algoritmu nasledovnými tvrdeniami: nech $\gamma : V \to V$ je permutácia V. Obrazom vrchola $v \in V$ podľa γ sa označuje v^γ. Pre zafarbený graf $G = (V, E, c)$ sa potom definuje zafarbený graf $G^\gamma = (V, \{\{v^\gamma, w^\gamma\} | \{v, w\} \in E\}, c^\gamma)$, kde funkcia $c^\gamma : V \to N$ je definovaná pre všetky vrcholy $v \in V$ ako $c^\gamma(v^\gamma) = c(v)$. Dva zafarbené grafy $G_1 = (V, E_1, c_1)$ a $G_2 = (V, E_2, c_2)$ sú izomorfné, ak existuje permutácia $\gamma : V \to V$ taká, že $G_1^\gamma = G_2$. Permutácia γ sa nazýva izomorfizmus G_1 na G_2.

Autormi *Bliss* sú *Tommi Junttila* a *Petteri Kaski*. *Bliss* pracuje aj s orientovanými grafmi. Aktuálna verzia *Bliss* je *0.72* pod licenciou *GNU GPL* [20].



Obr. 15: *Vrcholovo zafarbené grafy*

Na obrázku 15 vidíme dva izomorfné vrcholovo zafarbené grafy tromi farbami, kde jeden vrchol vyznačený čiernou výplňou má prvú farbu, dva vrcholy vyznačené bez výplne majú druhú farbu a jeden vrchol, ktorý je vyplnený krížikom je zafarbený treťou farbou.

A aký je v súčasnosti „state-of-art“? Okrem *nauty* a *bliss* poznáme aj program *VFlib*, ktorý je voľne šírený ako nástroj pre efektívne zisťovanie izomorfizmu. Desiatky ďalších voľne šíriteľných programov a knižníc možno vyhľadať a stiahnuť na internete. My použijeme knižnicu *iGraph*, ktorá nám ponúka aj *bliss* aj *VFlib*.

3.4 *iGraph*

Knižnica *iGraph* je *open source* projekt distribuovaný pod licenciou *GNU GPL* a je stále vo vývoji [16]. Knižnica *iGraph* je kolekcia algoritmov, ktorá obsahuje dostatočne veľké množstvo grafových algoritmov vrátane *Bliss* a funkcií pre operácie na grafoch, ako napríklad prehľadávanie grafu do hĺbky, zisťovanie štrukturálnych vlastností grafov, atď.

V dokumentácií udávajú časovú zložitosť pre každú funkciu, napríklad pre funkciu *igraph_isomorphic_bliss* udávajú exponenciálnu časovú zložitosť [17]. Okrem algoritmov obsahuje množstvo účinných dátových štruktúr pre reprezentáciu grafov a iné dátové štruktúry ako vektory dynamicky meniace veľkosť, zásobníky, obojsmerne lineárne fronty, haldy atď. Asi najbežnejšími používanými dátovými štruktúrami sú *igraph_t* a *igraph_vector_t*. *iGraph* poskytuje platformu pre vývoj a implementáciu grafových algoritmov pre developerov. Významnou vlastnosťou tejto knižnice je účinná manipulácia s veľkými grafmi, s obmedzením veľkosti fyzickej pamäti počítača, to znamená v súčasnosti niekoľko miliónov vrcholov a/alebo hrán. Autormi knižnice sú *Gábor Csárdi* a *Tamás Nepusz*. Knižnicu *iGraph* vyvíjajú od roku 2005 a aktuálna verzia je stará asi 3 mesiace a má označenie 0.6.3.

V programovej časti k tejto práci sme implementovali signované grafy do knižnice *iGraph* ako jednoduché grafy so znamienkovými atribútmi na hranách podľa názvu „signature“. Pre signované grafy sme implementovali algoritmy na zistenie, či sú signované grafy izomorfné, alebo ekvivalentné a izomorfné alebo ani ani, načítanie a uloženie signovaných grafov zo súboru/do súboru. Obidva algoritmy na ekvivalenciu aj izomorfizmus signovaných grafov volajú ďalšie dve implementované funkcie na kontrolu balansovanosti kružníc a prepínací algoritmus.

3.4.1 Balíček v jazyku R

Knižnica *iGraph* má väzbu aj na jazyk *R*. Jazyk *R* je prostredie pre štatistické výpočty a grafiku, pričom vie aj vykresliť jednoduché grafy. Je to GNU projekt, ktorý je podobný jazyku *S*. V rámci programovej časti k tejto práci sme robili aj pokusy s *iGraph*-om v jazyku *R*.

Práca s *iGraph*-om je jednoduchšia v *R* ako v C/C++ . Zvláštnou novinkou v *R* je celkové riešenie s dátovými štruktúrami. Dátová štruktúra n-rozmerný vektor je tu implemenovaná ako všeobecný priestor pre ukladanie dát. Napríklad graf je uložený v jednorozmernom vektore, kde každý prvok je *id* nejakého vrchola a hrany sú zakódované

ako disjunktné[10] dvojice susedných prvkov, pričom prvá hrana začína ako prvý a druhý prvok. Každý izomorfizmus je reprezentovaný ako vektor dĺžky n a viac izomorfizmov je tu uložených vo vektore, ktorého každý prvok je vektor s izomorfizmom.

Pre prácu v jazyku *R* slúži prostredie *Rstudio*, v ktorom je veľmi jednoduché pracovať. V prostredí *Rstudio* je knižnica *iGraph* priamo podporovaná ako „balík“, ktorý sa dá ľahko doinštalovať priamo v *Rstudiu* cez internet. Po prídaní „balíka“ *iGraph* je možné ihneď používať funkcie *iGraph*-u a zároveň je k dispozícií dokumentácia na všetky funkcie priamo v programe *Rstudio*. Tieto *iGraph*-ové funkcie sú veľmi prirodzene navrhnuté, ich kombináciou možno vykonávať s grafmi najrôznejšie operácie, či už zisťovanie vlastností grafu alebo ich modifikáciami. Najprv treba vytvoriť objekt - graf, ktorý sa dá jednoducho vytvoriť základnou metódou *graph.empty* alebo je možné použitím inej metódy *read.graph* nechať načítať graf ako zoznam hrán z externých textových súborov. Je praktickejšie zadávať grafy z externých súborov, ako upravovať zdrojový kód, v ktorom by bol graf uložený vo vektore napevno. V prostredí *Rstudio* je implementované aj jednoduché grafické zobrazenie grafov.

Tento rozširujúci modul *iGraph*-u do jazyka *R* je momentálne vo vývoji, rovnako tak aj pre *Python* a *Ruby*. Pre implementáciu algoritmov pre signované grafy sa bolo treba vrátiť k základnej verzií *iGraph*-u v jazyku C.

3.4.2 Knižnica v jazyku C

Celá knižnica *iGraph* je napísaná v jazyku *C*. Je to GNU projekt, ktorý sa kompiluje cez známy GNU GCC kompilátor. *iGraph* je ale väčší projekt a preto využíva program GNU make. Najprv je potrebné si knižnicu stiahnuť [16] zo stránky *iGraph*-u a „roztarovať“ príkazom napríklad: *tar xzf igraph-0.6.3.tar.gz* a následne prejsť do zložky igraph-0.6.3 príkazom: *cd igraph-0.6.3*.

Trojicou príkazov *./configure*, *make* a *make install* sa nainštaluje kompletná C knižnica. Prvým príkazom *./configure* sa nakonfiguruje a pripraví inštalácia knižnice a okrem iných súborov sa vygeneruje *Makefile*. Druhým príkazom *make* program GNU make prečíta vygenerovaný *Makefile* a „zbuilduje“ *iGraph*, to znamená skompiluje a zlinkuje množstvo zdrojových súborov a vytvorí statickú knižnicu *libigraph.a* a dynamickú knižnicu *libigraph.so.0*. Tretí príkaz *make install* prekopíruje statickú aj dynamickú knižnicu do systémového adresára */usr/local/lib*.

Pre začatie používania funkcií knižnice *iGraph*, stačí vo svojom programe pridať príkaz $\#include < igraph.h >$ na začiatku programu a povedať kompilátoru GCC, kde má hľadať knižnice a hlavičkové súbory, alebo si nastaviť systémové premenné. Potom

[10] n množín je disjunktných, ak neexistuje ani jeden taký prvok, ktorý by sa nachádzal vo viac ako jednej z n množín.

sú už k dispozícií všetky funkcie a dátové typy knižnice *iGraph*. Je veľmi nápomocné si vytvoriť k svojmu programu vlastný jednoduchý *Makefile* a pomocou príkazu *make* prekladať program. Teda pri každom preklade svojho programu stačí zadať jediný príkaz *make* a netreba vypisovať dlhé príkazy pre kompilátor GCC, kde má hľadať hlavičkové súbory a podobne, pretože to všetko je obsiahnuté v *Makefile*. Nasledujúci krátky program demonštruje základné použitie knižnice *iGraph*:

```
#include <igraph.h>
int main(void)
{
  igraph_t graph;
  igraph_empty(&graph, 5, IGRAPH_UNDIRECTED);
  igraph_destroy(&graph);
  return 0;
}
```

Tento triviálny program vytvorí neorientovaný jednoduchý graf s 5 izolovanými vrcholmi pomocou konštruktora *igraph_empty*, kde prvý parameter je pointer na zatiaľ neinicializovaný grafový objekt, druhý parameter je počet vrcholov a tretí parameter je zrejmý. Každý inicializovaný grafový objekt je potrebné na konci vymazať z pamäte pomocou deštruktora *igraph_destroy*.

Kapitola 4

4 Vlastný návrh a programová realizácia

K tejto práci sme navrhli algoritmus, ktorý rieši problém izomorfizmu signovaných grafov $G_1 = (H_1, \Sigma_1)$ a $G_2 = (H_2, \Sigma_2)$ za pomoci knižnice *iGraph*, vďaka ktorej máme k dispozícii izomorfizmy podkladových grafov H_1 a H_2 a súčasne množstvo ďalších užitočných funkcií ako napríklad *igraph_bfs*, ktorá prejde graf do šírky. Zadefinovali sme, čo je to izomorfizmus signovaných grafov tak, že musia byť splnené dve podmienky:

1. podkladové grafy H_1 a H_2 musia byť izomorfné,
2. pre každú kružnicu $C \in G_1$ musí byť aj jej obraz - kružnica $C' \in G_2$ rovnakej balansovanosti, t.j. C aj C' sú súčasne balansované alebo súčasne nebalansované.

V našom programe berieme izomorfizmus podkladových grafov H_1 a H_2 ako „nejakú“ bijekciu množín vrcholov V_1 do V_2, nad ktorou pracuje náš algoritmus na overenie balansovanosti.

4.1 Hlavný program

Myšlienka programu, ktorý rozhodne, či sú alebo nie sú dva ľubovoľné zadané signované grafy $G_1 = (H_1, \Sigma_1)$ a $G_2 = (H_2, \Sigma_2)$ izomorfné je nasledovná: Postupne pre každý izomorfizmus φ_i podkladových grafov H_1 a H_2 je treba overiť podľa nejakej kostry, či každá fundamentálna kružnica má svoj obraz rovnakej balansovanosti. Ak pre nejaké φ_i a nejakú kostru T bude mať každá fundamentálna kružnica svoj obraz rovnakej balansovanosti, potom sú signované grafy izomorfné. Ak pre žiadne φ_i nebudú mať podľa nejakej kostry všetky fundamentálne kružnice svoje obrazy rovnakej balansovanosti, potom signované grafy nie sú izomorfné. Po zistení, že sú signované grafy izomorfné sa zavolá funkcia, ktorá poprepína vrcholy na prvom grafe a vráti informáciu o tom, ktoré vrcholy na prvom grafe treba prepnúť aby sa G_1 zmenil na G_2 pre dané φ.

V hlavnom programe *main* voláme funkcie pre signované grafy, ktoré sú zabalené v module *signed.c*, ktorý sme implementovali do knižnice *iGraph*. Program *main* číta na vstupe signované grafy uložené v textových súboroch a výstupom sú výpisy na obrazovku alebo pri spustení s parametrom 1 uloží výsledok výpočtu do súboru. Program je priložený v prílohe k práci a na jeho spustenie je potrebné mať nainštalovanú knižnicu *iGraph*.

4.2 Implementácia signovaných grafov do knižnice *iGraph*

Na to aby bolo možné pracovať so signovanými grafmi s knižnicou *iGraph*, bolo potrebné najprv implementovať signované grafy. Knižnica *iGraph* umožňuje pridať jednoduchým grafom rôzne celočíselné alebo reťazcové atribúty na vrcholy, hrany alebo aj na celý graf.

Signované grafy sme do knižnice *iGraph* implementovali tak, že sú to jednoduché grafové objekty typu *igraph_t* so znamienkovými atribútmi podľa názvu „signature“, teda každá hrana má podľa svojho *id* hrany priradené znamienko. Záporné hrany majú priradené atribúty 1 a kladné hrany majú priradené atribúty 0. Je prirodzené mať takto zakódované znamienka, pretože pri počítaní balansovanosti fundamentálnej kružnice stačí sčítať 0 a 1 na jej hranách a ak je výsledok párne číslo, potom je kružnica balansovaná.

Signované grafy by mohli mať aj iné atribúty na hranách, napríklad by boli hranovo ohodnotené nejakými váhami, avšak k atribútom sa pristupuje cez názov atribútov. Znamienka majú názov „signature“ a pretože *iGraph* dovoľuje mať aj viac druhov atribútov rozlíšiteľných podľa názvu, takouto implementáciou signovaných grafov sme neubrali možnosť rozšírenia o ďalšie atribúty na hrany. Teda signované grafy sú implementované veľmi konzervatívnym spôsobom.

Pre implementáciu signovaných grafov bolo samozrejme potrebné vytvoriť funkciu, ktorá ich takto inicializuje a ďalšie funkcie, ktoré vedia pracovať s takto implementovanými signovanými grafmi, pretože *iGraph* nepozná signované grafy. Všetky tieto funkcie sú zabalené v implementovanom module pre signované grafy, ktorý je vysvetlený v nasledujúcej časti.

4.3 Modul pre signované grafy do knižnice *iGraph*

Vytvorili sme modul *signed.c* pre signované grafy do knižnice *iGraph* tak, aby knižnica *iGraph* nebola týmto modulom nijako modifikovaná. Teda je možné modul prilinkovať s knižnicou bez toho, aby boli vykonané nejaké zmeny v niektorom zdrojovom súbore knižnice *iGraph*. Tento modul obsahuje niekoľko funkcií pre signované grafy, z

ktorých je najdôležitejšia funkcia *igraph_signed_graph_isomorphic*, ktorá rozhodne, či sú ľubovoľné dva zadané signované grafy izomorfné.

Rozhrania funkcií implementovaného modulu *signed.c* sú deklarované v hlavičko-vom súbore *signed.h* a vyzerajú nasledovne:

```
int igraph_signed_initialize();
int igraph_signed_deinitialize();
int igraph_load_signed_graph(igraph_t *g, FILE *f);
int igraph_save_signed_graph(igraph_t *g, FILE *f);

int igraph_signed_graph_equivalent(igraph_t *g1, igraph_t *g2,
igraph_vector_t *sw, igraph_bool_t *iso);
int igraph_signed_graph_isomorphic(igraph_t *g1, igraph_t *g2,
igraph_vector_t *img, igraph_vector_t *sw, igraph_bool_t *iso);

int igraph_signed_graph_fundamental_circles(igraph_t *g1, igraph_t *g2,
igraph_vector_t *bij, igraph_bool_t *iso);
int igraph_signed_graph_switch(igraph_t *g1, igraph_t *g2,
igraph_vector_t *bij, igraph_vector_t *sw, igraph_bool_t *iso);
```

Presný popis hlavičiek implementovaných funkcií:

`int igraph_signed_initialize()`

- Funkcia bez parametrov, je prerekvizitou pre všetky ďalšie implementované funkcie. Zapína všetky atribúty pre jednoduché grafy - na vrcholy, na hrany, na graf (ktoré sú v knižnici *iGraph* „defaultne“ vypnuté), aby bolo možné rozšírenie pre signované grafy.

`int igraph_signed_deinitialize()`

- Funkcia bez parametrov, ktorá nemá žiadnu funkciu a jej význam je provizorný ako komplement ku *igraph_signed_initialize()*

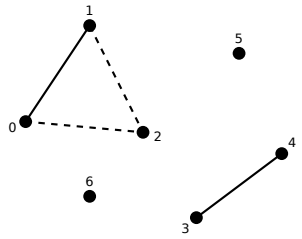
`int igraph_load_signed_graph(igraph_t *g, FILE *f)`

- Načíta signovaný graf priamo zo súboru, inicializuje grafový objekt a vytvorí z neho signovaný graf pridaním znamienok ako atribúty na hrany podľa mena „signature“. Má dva parametre: *g - pointer na zatiaľ neinicializovaný grafový objekt typu *igraph_t*, *f - pointer na stream typu súbor otvorený na čítanie. Prerekvizity tejto funkcie sú:

existujúci súbor so správnym formátom signovaného grafu, obidva parametre musia byť korektné, teda pointre nastavené na príslušne adresy objektov. Predpísaný formát signovaného grafu v súbore:

```
7
0 1 0
0 2 1
2 1 1
3 4 0
```

V prvom riadku je jedno celé nezáporné číslo udávajúce počet vrcholov z dôvodu, že signovaný graf môže byť aj nesúvislý a mať izolované vrcholy. V každom ďalšom riadku je jedna hrana ako trojica celých nezáporných čísel, pričom prvé dve čísla sú *id* vrcholov tvoriacich hranu, tretie číslo je 0 alebo 1 pre kladnú, respektíve zápornú hranu. Predchádzajúci graf takto zakódovaný v súbore vyzerá nasledovne:



Obr. 16: *Nesúvislý signovaný graf*

```
int igraph_save_signed_graph(igraph_t *g, FILE *f)
```

- Uloží existujúci signovaný graf do súboru podľa predpísaného formátu. Má dva parametre: *g - pointer na inicializovaný signovaný graf, *f - pointer na stream typu súbor otvorený iba na zápis. Ak súbor neexistuje, vytvorí ho a zapíše do neho signovaný graf, ak existuje, vymaže ho a zapíše do neho signovaný graf.

```
int igraph_signed_graph_equivalent(igraph_t *g1, igraph_t *g2,
igraph_vector_t *sw, igraph_bool_t *iso)
```

- Táto funkcia rozhodne, či sú ľubovoľné dva zadané signované grafy ekvivalentné. Prvé dva parametre *g1 a *g2 sú pointre na dva inicializované signované grafy, tretí parameter *sw je pointer na inicializovaný nulový vektor sw dĺžky $n = |V(G_1)| = |V(G_2)|$,

do ktorého funkcia uloží informáciu o prepnutých vrcholoch G_1. Ak na príslušnom indexe i vektora sw sa po skončení nachádza hodnota 1, znamená to, že bol prepnutý vrchol $u \in G_1$, pričom *id* vrchola u je i. Posledný parameter *iso je pointer na logickú premennú typu *igraph_bool_t*, ktorá nadobúda hodnotu 1 (True) vtedy a iba vtedy, ak sú G_1 a G_1 ekvivalentné.

```
int igraph_signed_graph_isomorphic(igraph_t *g1, igraph_t *g2,
igraph_vector_t *img, igraph_vector_t *sw, igraph_bool_t *iso)
```

- Hlavná funkcia implementovaného modulu, rozhodne či sú ľubovoľné dva zadané signované grafy izomorfné. Prvé dva parametre *g1 a *g2 sú pointre na inicializované signované grafy. Tretí parameter *img je pointer na inicializovaný nulový vektor dĺžky n, ktorý bude obsahovať izomorfizmus podkladových grafov, ak sú signované grafy izomorfné, inak zostane nulový. Izomorfizmus podkladových grafov je zakódovaný jednoduchým spôsobom tak, že *id* vrcholov G_1 sú indexy vektora img a hodnoty sú ich obrazy. Štvrtý parameter *sw je pointer na inicializovaný nulový vektor dĺžky n, do ktorého funkcia uloží informáciu o prepnutých vrcholoch podobne ako v predchádzajúcej funkcii. Posledný parameter je podobne ako v predchádzajúcej funkcii pointer na logickú premennú, ktorá je nastavená na hodnotu 1 (True) vtedy a iba vtedy, ak sú G_1 a G_2 izomorfné.

```
int igraph_signed_graph_fundamental_circles(igraph_t *g1, igraph_t *g2,
igraph_vector_t *bij, igraph_bool_t *iso)
```

- Toto je funkcia, ktorá pre každú fundamentálnu kružnicu C na G_1 overí, či ona aj jej obraz C' na G_2 majú rovnakú paritu záporných hrán. Táto funkcia je volaná z oboch funkcií *igraph_signed_graph_equivalence* a *igraph_signed_graph_isomorphic*. Prvé dva parametre *g1 a *g2 sú pointre na inicializované signované grafy, ďalší vstupný parameter *bij je pointer na vektor bij, ktorý obsahuje alebo identitu, alebo izomorfizmus podkladových grafov, podľa toho či je táto funkcia volaná z funkcie ekvivalencie alebo izomorfizmu signovaných grafov. Posledným parametrom je pointer *iso na logickú premennú iso, ktorá po skončení funkcie bude obsahovať 1 (True) vtedy a iba vtedy, ak pre žiadnu fundamentálnu kružnicu C na G_1 nezistí nerovnakú paritu záporných hrán s jej obrazom C' na G_2.

```
int igraph_signed_graph_switch(igraph_t *g1, igraph_t *g2,
igraph_vector_t *bij, igraph_vector_t *sw, igraph_bool_t *iso)
```

- Funkcia, ktorá poprepína vrcholy na G_1. Táto funkcia je volaná z oboch funkcií *igraph_signed_graph_equivalence* a *igraph_signed_graph_isomorphic* a je volaná vtedy a

iba vtedy, ak funkcia *igraph_signed_graph_fundamental_circles* vracia True cez parameter iso. Prvé dva parametre *g1 a *g2 sú pointre na inicializované signované grafy. Tretí parameter *bij je pointer na vektor bij obsahujúci opäť alebo izomorfizmus podkladových grafov, alebo identitu v závislosti na tom, z akej funkcie bola táto funkcia volaná. Posledné dva parametre sú výstupné a pointer *sw ukazuje na inicializovaný nulový vektor dĺžky n, v ktorom bude informácia o prepnutých vrcholoch. Teda vrchol $u \in V(G_1)$ s *id* i bol prepnutý vtedy a iba vtedy, ak na indexe i vektora sw sa nachádza 1. Posledný parameter *iso ukazuje na logickú premennú iso, ktorá nadobudne hodnotu 0 (False) iba ak neexistuje prepnutie.

4.4 Funkcia *igraph_signed_graph_isomorphic*

Funkcia *igraph_signed_graph_isomorphic* implementovaného modulu *signed.c* zodpovie na hlavnú otázku tejto práce, teda izomorfizmu signovaných grafov. Popis parametrov, vstupy, výstupy sú popísané v predchádzajúcej podkapitole. Funkcia očakáva korektný vstup. Uvedieme pseudokód a popíšeme zaujímavé časti.

Pseudoalgoritmus:

```
(1) nastav výstupnú logickú premennú iso na 0 (False)
(2) zavolaj iGrafovú funkciu igraph_get_isomorphisms_vf2,
    ktorá vráti všetky izomorfizmy podkladových grafov
    - ak nenájde ani jeden izomorfizmus podkladových grafov,
    koniec hlavnej funkcie, vektory img, sw zostanú nulové a
    logická premenná iso zostane nastavená na 0 (False)
(3) pre každý izomorfizmus podkladových grafov vykonaj:
  (3.1) zavolaj funkciu igraph_signed_graph_fundamental_circles
        - ak igraph_signed_graph_fundamental_circles vráti
          (False), pokračuj krokom (3) nasledujúcou iteráciou
        - ak igraph_signed_graph_fundamental_circles vráti
          (True), pokračuj
    (3.1.1) zavolaj funkciu igraph_signed_graph_switch,
          - do vektora img ulož izomorfizmus podkladových grafov
            pre ktorý igraph_signed_graph_fundamental_circles
            vrátila (True),
          - do vektora sw ulož informáciu o prepnutých vrcholoch
            ktorú vráti funkcia igraph_signed_graph_switch
          - logickú premennú iso nastav na 1 (True)
```

```
          - koniec hlavnej funkcie.
(4) ak pre žiadny izomorfizmus funkcia
    igraph_signed_graph_fundamental_circles nevráti 1 (True),
    koniec hlavnej funkcie, vektory img, sw zostanú nulové a
    logická premenná iso zostane nastavená na 0 (False).
```

(1) Časovo najnáročnejšiu úlohu v tejto funkcií robí iGrafová funkcia *igraph_get_isomorphisms_vf2*, pretože nám nájde všetky izomorfizmy podkladových grafov. V dokumentácií je uvedaná časová zložitosť exponenciálna. Ostatné operácie vyžadujú menej času na výpočet a ďalšie dve implementované funkcie *igraph_signed_graph_fundamental_circles* a *igraph_signed_graph_switch* potrebujú menej času na výpočet ako *igraph_get_isomorphisms_vf2*. Teda hlavná funkcia *igraph_signed_graph_isomorphic* má exponenciálnu časovú zložitosť.

4.5 Algoritmus balansovanosti

Algoritmus je implementovaný vo funkci *igraph_signed_graph_fundamental_circles*. Je to dôležitá overovacia funkcia, ktorá vráti True ak nenájde v G_1 kružnicu C, ktorá by bola balansovaná a jej obraz C' v G_2 nebalansovaná alebo naopak (podľa bijekcie vrcholov uloženej na vstupe vo vektore bij nad ktorým funkcia pracuje).

4.5.1 Činnosť algoritmu balansovanosti

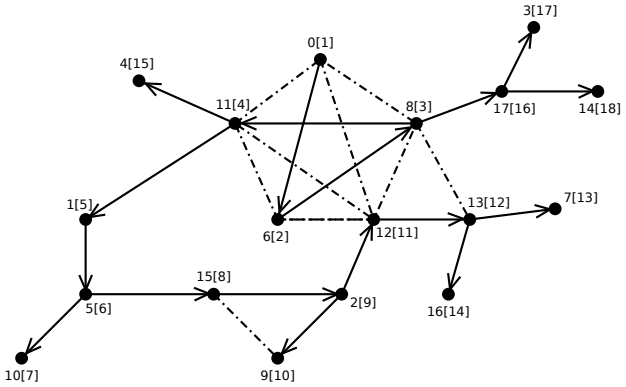
Funkcia *igraph_signed_graph_fundamental_circles* má asi 100 riadkov kódu a preto uvedieme iba stručný pseudokód a popíšeme hlavné, resp. zaujímavé časti kódu.

```
...
/*deklarácia a inicializácia premenných*/
...
/*deklarácia a inicializácia vektorov*/
...
(1) zavolaj iGrafovú funkciu igraph_dfs, ktorá prejde graf g1 (1)
    do hĺbky, vráti postupnosť objavených vrcholov a pre každý
    vrchol informáciu o tom, z ktorého vrchola bol objavený
(2) pre každú hranu e grafu g1 urob nasledovné:
  (2.1) zisti id koncových vrcholov e
  (2.2) zisti či e je mimokostrová hrana
        - ak nie je mimokostrová, pokračuj nasledujúcou
          iteráciou v bode (2)
```

```
        - ak je mimokostrová, pokračuj:
    (2.2.1) zisti, ktorý vrchol hrany e = {u,v} bol objavený neskôr,
            nech je to vrchol u
    (2.2.2) prejdi spätne z vrchola u do vrchola v (2)
            po kružnici C, sčítaj jej znamienka a sučasne
            znamienka C' na g2 podľa bijekcie vrcholov
            uloženej vo vektore bij
    (2.2.3) pričítaj k počítadlám znamienka na hrane e
    (2.2.4) ak počet záporných hrán na C je párny a počet
            záporných hrán na C' nepárny (alebo naopak),
            - nastav logickú premennú iso na False
            - koniec hlavnej funkcie
            inak pokračuj d'alšou iteráciou v bode (2).
...
/*ak sa po ukončení hlavného cyklu v bode (2) dostane sem,
znamená to, že neobjavil žiadnu fundamentálnu kružnicu C,
ktorej obraz by bol inej balansovanosti.*/
...
/*deinicializácia vektorov*/
(3) nastav logickú premennú iso hlavnej funkcie na True
(4) koniec funkcie
```

(1) - *iGraph*-ová funkcia *igraph_dfs* prehľadá graf g1 do hĺbky podľa nejakej kostry T a to hlavná funkcia využije k nájdeniu mimokostrových hrán. Funkcia *igraph_dfs* má 11 parametrov a nás hlavne zaujíma 5. a 7. parameter - vektory **order** a **father**. Vektor order obsahuje poradie objavených vrcholov a vektor father obsahuje informáciu o tom, ktorý vrchol bol z ktorého vrchola objavený. Zaujmavý je 2. a 4. parameter. Druhý parameter je 0 - prehľadávanie sa začína v „koreni“ - vrchol s *id* 0. Štvrtý parameter 1 - zapína prehľadávanie pre nesúvislé grafy.

(2) - Ak máme mimokostrovú hranu $e = \{u, v\}$ znamená to, že u nebol objavený z v a súčasne v nebol objavený z u. Keďže jeden z nich bol objavený neskôr povedzme u, potom z neho existuje tzv. spätná cesta do v podľa vektora father. Nemôže sa totiž stať, že by sa z u nedostal do v po spätnej ceste. Aby toto nastalo, musel by mať koreň stupeň aspoň 2 a musel by sa vrátiť z jednej svojej najhbšej vetvy do koreňa, z neho ísť do druhej svojej vetvy a z nej nájsť mimokostrovú hranu, ktorá siaha do vetvy, z ktorej sa už raz vrátil. Toto nenastane, pretože ak sa z jednej vetvy vráti do koreňa a prejde do inej vetvy, potom by prienik týchto vetiev bol iba koreň.



Obr. 17: *Prechádzka grafu do hĺbky*

Na obrázku 17 je jednoduchý graf na 18 vrcholoch (označených od 0 po 17) a 26 hranách. Deväť mimokostrových hrán je vyznačených bodkočiarkovanými čiarami. Šípka na plne vyznačenej hrane naznačuje, ktorý vrchol hrany bol objavený z ktorého. Vrcholy sú označené číslami naľavo od hranatých zátvoriek a čísla v hranatých zátvorkách hovoria o poradí objavenia. Algoritmus nájde mimokostorvú hranu $e = \{u, v\}$ podľa toho, že hrana e nemá šípku. Fundamentálnu kružnicu prejde po hranách podľa šípiek spätne z vrchola s väčším číslom v zátvorke do vrchola s menším číslom v zátvorke vrátane hrany $\{u, v\}$.

Vektor **order** vyzerá nasledovne: 0 6 8 11 1 5 10 15 2 9 12 13 7 16 4 17 3 14 - táto postupnosť znamená, že prvý bol objavený vrchol s označením 0, druhý bol objavený vrchol s označením 6, ..., posledný bol objavený vrchol s označením 14. Vektor **father** vráti: -1 11 15 17 11 1 0 13 6 2 5 8 2 12 17 5 13 8 - indexy vektora father sú vrcholy prehľadávaného grafu a táto postupnosť znamená, že koreň (index 0 = označenie 0) nebol objavený zo žiadneho vrchola, preto má -1. Vrchol 1 bol objavený z vrchola 11, vrchol 2 bol objavený z vrchola 15, ..., až vrchol 17 bol objavený z vrchola 8.

4.5.2 Časová zložitosť algoritmu balansovanosti

Na začiatku je inicializovaných niekoľko vektorov konštruktorom *igraph_vector_init* jeho časová zložitosť $O(|V|)$ je udávaná v závislosti na operačnom systéme podľa toho aký „čas“ potrebuje na alokáciu n elementov. Funkcia *igraph_dfs* má podľa dokumentácie časovú zložitosť $O(|V| + |E|)$ - lineárnu podľa počtu vrcholov a hrán. Zistenie koncových vrcholov sa vykoná funkciou *igraph_edge* v čase $O(1)$ a v rovnakom čase aj či je, respektíve nie je, mimokostrová. Pre každú mimokostrovú hranu zistí, ktorý vrchol bol objavený z ktorého, najviac v čase $O(|E|.|V|)$. Pri spätnej prechádzke po fundamentálnej kružnici dvakrát zisťuje *id* hrany po ktorej ide a dvakrát znamienko na hrane

po ktorej ide. Zistenie *id* hrany sa realizuje v čase $O(log(d_1) + log(d_2))$, kde d_1 a d_2 sú stupne vrcholov hrany. Znamienko prečíta z grafového objektu funkciou EAN v čase $O(|E|)$. Ostatné operácie bežia v podobných časoch. Celkový čas funkcie je lineárny $O(|V| + |E|)$ vzhľadom na počet hrán, až kvadratický vzhľadom na počet vrcholov, lebo počet hrán môže byť rádovo až $|V|^2$.

4.6 Prepínací algoritmus

Prepínací algoritmus implementovaný vo funkcií *igraph_signed_graph_switch* poprepína vrcholy na G_1 a vráti informáciu o tom, ktoré vrcholy boli prepnuté na G_1.

4.6.1 Činnosť prepínacieho algoritmu

Táto funkcia je dlhá približne 100 riadkov kódu a preto uvedieme jej stručný pseudokód a zaujímavé časti sú očíslované v zátvorkách napravo a detailnejšie rozobrané pod textom.

```
...
/*deklarácia a inicializácia premenných*/
...
/*inicializácia vektorov*/
...
(1) skopíruj znamienka z g1 do vektora signs - v ktorom
    sa pri prepínaní budú meniť hodnoty
(2) zavolaj funkciu igraph_bfs - prehľadá g1 do šírky (1)
(3) pre každý prvok vektora order,
    okrem prvého prvku vykonaj nasledovné:
  (3.1) do premennej u ulož prvok vektora order -
        id vrchola podľa poradia objavenia od koreňa,
        začínajúc druhým prvkom
  (3.2) prejdi každý vrchol v, ktorý bol skôr objavený ako
        vrchol u a skontroluj, či tvoria hranu e (2)
        - ak {u,v} je hrana e, porovnaj znamienko na e
          v g1 a znamienko na obraze e' v g2
          - ak {u,v} a {u',v'} maju rovnaké znamienka,
            zvýš počítadlo P1 pre rovnaké hrany o 1
          - ak {u,v} a {u',v'} majú iné znamienka,
            zvýš počítadlo P2 pre iné hrany o 1
        - ak {u,v} nie je hrana e, pokračuj d'alším
```

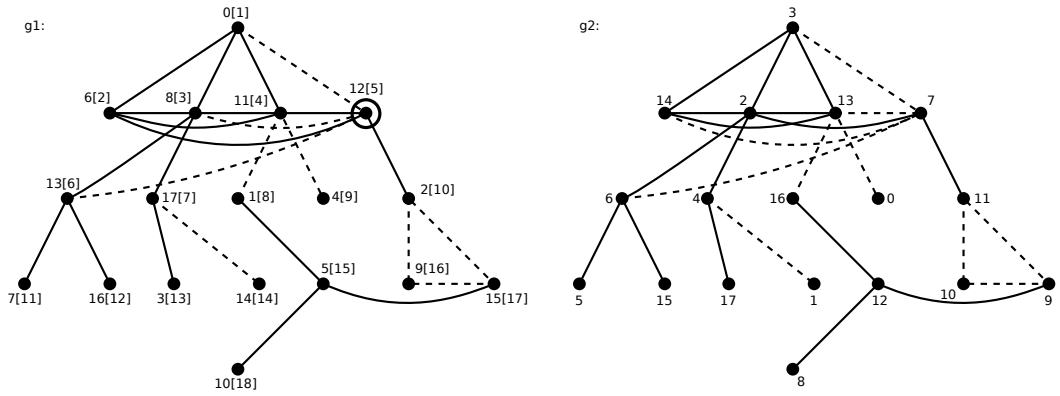
```
          vrcholom v kroku (3.2)
  (3.3) ak P1 = 0 a P2 > 0 prepni vrchol u (3)
  (3.4) ak P1 > 0 a P2 > 0 koniec - neexistuje prepnutie (4)
        - nastav iso na False
        - koniec hlavnej funkcie
(4) ak prejde všetky prvky vektora order - vrcholy g1
    niektoré mohli aj nemuseli byť prepnuté,
    - ulož do parametra sw hlavnej funkcie informáciu o
      prepnutých vrcholoch
    - nastav iso na True
    - koniec hlavnej funkcie
```

(1) - Prehľadávanie do šírky pomocou *iGraph*-ovej funkcie *igraph_bfs*, korá má 14 parametrov nám vráti opäť vektor **order** - poradie objavených vrcholov. V tomto vektore máme zároveň zakódovanú informáciu o vzdialenosti vrcholov od koreňa, z ktorého sa začína prehľadávanie. Taktiež sme povolili prehľadávanie aj pre nesúvislé grafy.

(2) - Pre každý vrchol u vektora order skontroluje, či vrchol v, ktorý je skôr v poradí vektora order ako u tvorí hranu $e = \{u, v\}$. Každý vrchol je v nejakej vzdialenosti od koreňa. Okrem koreňa ide vyberať postupne vrcholy podľa poradia order, lebo pri prehľadávaní do šírky objaví najprv vrcholy vo vzdialenosti 1 od koreňa, potom vrcholy vo vzdialenosti 2 od koreňa atď. Ak je vrchol u vo vzdialenosti k od koreňa, skontroluje iba hrany s ním incidentné, ktoré končia vo vrcholoch vo vzdialenosti k alebo $k-1$ a súčasne boli objavené skôr ako u.

(3) - Vrchol u vo vzdialenosti k môže byť prepnutý iba vtedy, ak sú všetky hrany (končiace vo vzdialenosti k a $k-1$, vo vrcholoch skôr objavených) s ním incidentné medzi $g1$ a $g2$ opačné. Ak by bola hrana e napr. kladná a jej obraz e' tiež kladná a ostatné opačné, potom by prepnutím u na $g1$ bola hrana e záporná a na $g2$ by zostala e' kladná a ostatné by boli v poriadku.

(4) - Ak ide z u (vo vzdialenosti k) l hrán do l vrcholov vo vzdialenosti k a $k-1$, musia medzi $g1$ a $g2$ mať všetky opačné znamienka, aby bolo možné vrchol u prepnúť. Teda ak i-ta hrana $\{u, v\}$ je kladná (záporná), tak $\{\varphi(u), \varphi(v)\}$ je záporná (kladná).



Obr. 18: *Signované grafy zavesené na koreni*

Na obrázku 18 máme ten istý graf ako je na obrázku 17 ale nakreslený tak, že je zavesený na koreni (vzdialenosť 0) a ostatné vrcholy sú vo vzdialenosti od koreňa 1,2,3 a 4. Naľavo je $g1$ a napravo $g2$, prepínací algoritmus prepína vrcholy na $g1$ podľa izomorfizmu φ, kde $\varphi(0) = 3, \varphi(1) = 16, \varphi(2) = 11, \varphi(3) = 17, \varphi(4) = 0, \varphi(5) = 12, \varphi(6) = 14, \varphi(7) = 5, \varphi(8) = 2, \varphi(9) = 10, \varphi(10) = 8, \varphi(11) = 13, \varphi(12) = 7, \varphi(13) = 6, \varphi(14) = 1, \varphi(15) = 9, \varphi(16) = 15, \varphi(17) = 4$. Na $g1$ majú vrcholy v hranatých zátvorkách poradie, v akom boli objavené pri BFS.

Ak by u bol zakrúžkovaný vrchol, nebolo by ho možné prepnúť, pretože prepnutím by hrany $\{12, 6\}$, $\{12, 8\}$ a $\{12, 11\}$ na $g1$ boli rovnaké ako ich obrazy $\{7, 14\}$, $\{7, 2\}$, $\{7, 13\}$ na $g2$, ale hrana $\{12, 0\}$ na $g1$ by nebola rovnaká ako jej obraz $\{7, 3\}$ na $g2$. Toto ale nenastane, pretože funkcia *igraph_signed_graph_switch* bola volaná po overovacej funkcií *igraph_signed_graph_fundamental_circles*.

4.6.2 Časová zložitosť prepínacieho algoritmu

Na začiatku sa inicializujú vektory, každý v čase závisejúcom od operačného systému na alokáciu $O(|V|)$ elementov. Na prekopírovanie znamienok do vektora pre prepínanie je potrebný čas $O(|E|)$. Funkcia *igraph_bfs* má tiež časovú zložitosť $O(|V| + |E|)$ ako *igraph_dfs*. Dva vnorené cykly bežia v čase $O(|V| \cdot (|V| - 1))$. V nich prebiehajú operácie, ktoré si vyžadujú čas $O(log(d1) + log(d2))$, kde *d1* a *d2* sú stupne koncových vrcholov nejakej hrany. Zvyšné operácie si vyžadujú čas $O(1)$. Celková zložitosť bude $O(|V|^3)$.

Funkcie, ktoré sme naprogramovali pracujú v polynomiálnom čase a ak by existoval polynomiálny algoritmus na izomorfizmus jednoduchých grafov, potom by aj náš algoritmus pre signované grafy pracoval v polynomiálnom čase.

Kapitola 5

5 Portovanie do Windows

Hlavný program *main*, ktorý využíva modul *signed.c* implementovaný do knižnice *iGraph* sme urobili na linuxovej distribúcií Ubuntu 12.04 LTS. Neskôr bolo potrebné mať program *main* aj pre operačný systémý *Windows*. Keďže program *main* je napísaný v jazyku C a skompilovaný pre unix, bolo potrebné ho skompilovať pre Windows iným spôsobom.

Keďže pri kompilácií programu *main* sa linkuje modul *signed.c* s unixovou verziou *iGraph*-ovej dynamickej knižnice *libigraph.so*.0, bolo potrebné mať aj Windowsovú verziu *iGraph*-ovej dynamickej knižnice. *iGraph* je GNU projekt a na to, aby sme dostali jeho „zbuildovaním“ windowsovú dynamickú knižnicu libigraph-0.dll, použili sme na to *open source* nástroj *MinGW*[11].

Po nainštalovaní MinGW [21], sme cez program *MSYS*[12], ktorý sa inštaluje spolu s MinGW, trojicou príkazov $./configure$, $make\ CFLAGS = -DWin32$ a $make\ install$ „zbuildovali“ *iGraph*, čoho produktom sme získali dynamickú knižnicu ako súbor libigraph-0.dll - verziu pre Windows. Teraz sme už mohli jednoducho skompilovať program *main* pre Windows tak, že sme si upravili *Makefile* ku *main* a príkazom *make* v prostredí MSYS zlinkovali libigraph-0.dll s modulom signed.c a výsledkom sme dostali súbor main.exe. Je to hlavný program, ktorý číta na vstupe signované grafy zo súborov a výsledok výpočtu (či sú izomorfné) vypíše na obrazovku.

5.1 Aplikácia pre grafické zadávanie signovaných grafov

K hlavnému programu *main.exe*, ktorý načítava signované grafy zo súborov a výstupom je výpis na obrazovku sme urobili ako „nadstavbu“ - *GUI*[13] - oknovú aplikáciu

[11] Minimalist GNU for Windows - port GNU nástrojov GCC, make, assembler, linker, ... pre operačný systém Windows.

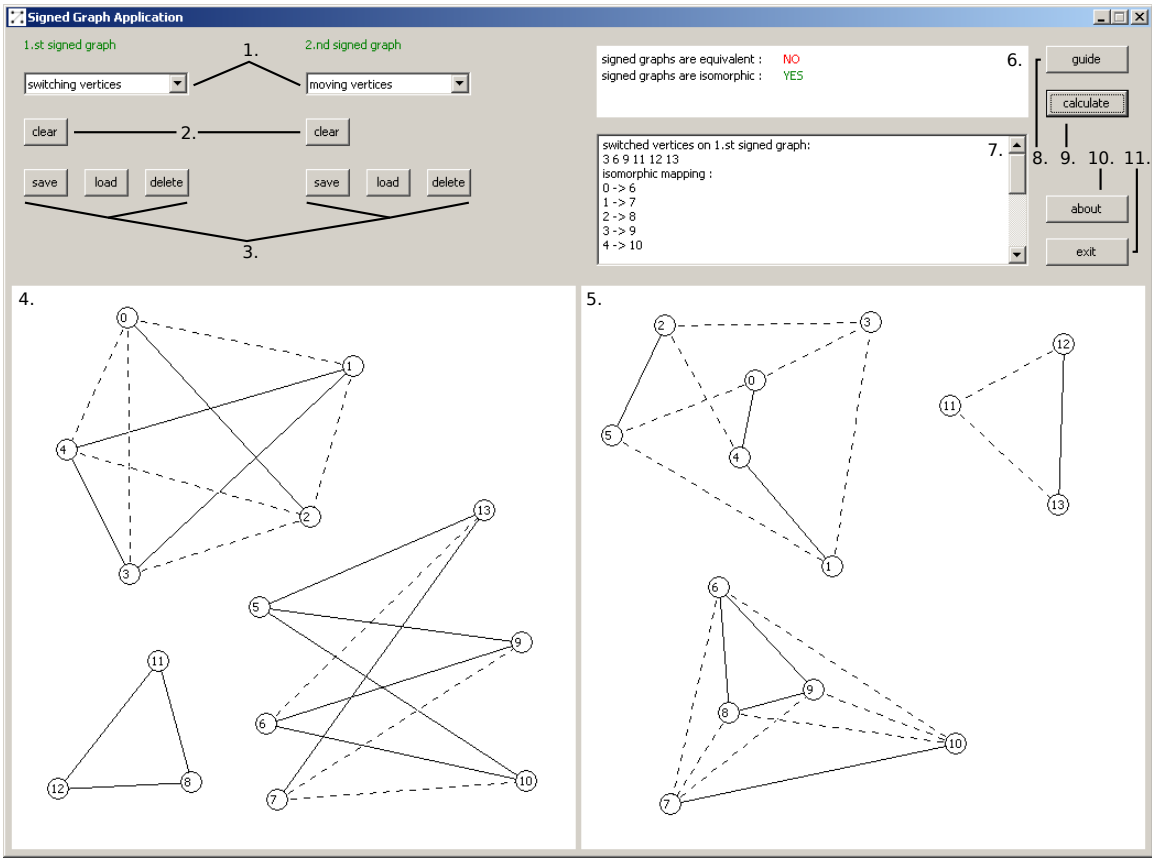
[12] Minimal SYStem - je to shell príkazový riadok a interpréter, je to alternatíva ku windowsovému cmd.exe

[13] GUI - graphics user interface je program, ktorý má užívateľsky príjemné grafické vstupno - výstupné rozhranie.

pre operačný systém Windows. Signované grafy sa v ňom zadávajú graficky pomocou myši a výstupom sú textové výpisy, ako napríklad, či sú takto zadané dva signované grafy izomorfné.

5.2 Popis, časti okna aplikácie

Na obrázku 19 vidíme ukážku aplikácie.



Obr. 19: *GUI aplikácia pre signované grafy*

Jednotlivé časti okná sú:

1. Výber operácie pre prvý (druhý) signovaný graf. Je to vysúvacie menu so 7 položkami, pričom vybraná je vždy len jedna položka - operácia:
- adding vertices - klikaním do plochy sa vytvárajú nové vrcholy
- adding positive edges - pridá kladnú hranu kliknutím na dva rôzne vrcholy
- adding negative edges - pridá zápornú hranu kliknutím na dva rôzne vrcholy
- deleting vertices - klikaním na vrcholy ich vymaže spolu s incidentnými hranami
- deleting edges - vymaže hranu kliknutím na jej incidentné vrcholy
- moving vertices - posúvanie vrcholov, úprava

- switching vertices - klikaním na vrcholy ich prepína

2. Tlačidlo **clear** pre prvý (druhý) signovaný graf, vymaže aktuálny signovaný graf z obrazovky a dátovej štruktúry v programe.

3. Tri tlačidlá (jedna trojica pre prvý, druhá trojica pre druhý signovaný graf):
- **save** - Uloženie signovaného grafu do databázy podľa názvu. Po kliknutí na toto tlačidlo sa objaví malé okno so vstupným textovým poľom do ktorého používateľ zadá názov z klávesnice. Potvrdením (kliknutím na tlačidlo OK) sa uloží aktuálne vykreslený signovaný graf. Pri pridaní nového grafu do databázy sa skontroluje správnosť nového názvu (iný ako predošlé uložené, počet znakov aspoň 5 a najviac 25, bez medzier).
- **load** - Načítanie signovaného grafu z databázy podľa názvu. Po kliknutí na toto tlačidlo sa objaví malé okno, v ktorom budú zobrazené položky s názvami uložených grafov. Po vybratí (kliknutím na položku) a potvrdení tlačidlom OK sa z databázy načíta vybraný signovaný graf do dátovej štruktúry a vykreslí sa presne tak, ako bol vykreslený pri ukladaní.
- **delete** - Vymazanie uloženého signovaného grafu z databázy. Po kliknutí na toto tlačidlo sa objaví malé okno, v ktorom budú zobrazené položky s názvami uložených grafov. Po vybratí (kliknutím na položku) a potvrdení tlačidlo OK sa vymaže z databázy vybraný signovaný graf.

4. a 5. Vstupné a zobrazovacie plochy prvého (druhého) signovaného grafu, do ktorých sa klikaním (kliknutím a ťahaním) myšou podľa výberu operácie z 1. zadávajú, modifikujú, upravujú signované grafy.

6. a 7. Textové plochy, vypisujú sa v nej komentáre (napríklad ak používatel uloží prvý (druhý) signovaný graf do databázy pod zadaným názvom).

8. Tlačidlo **guide** - podrobnejšia používateľská príručka o funkciách aplikácie.

9. Tlačidlo **calculate** - vypočíta, či sú signované grafy ekvivalentné a izomorfné alebo izomorfné alebo ani ani a výsledok vypíše do textovej plochy 6, informáciu o prepnutých vrcholoch a izomorfizmus podkladových grafov do textovej plochy 7.

10. Tlačidlo **about** - info o programe (na čo program slúži, autor, rok, . . .).

11. Tlačidlo **exit** - ukončenie aplikácie.

Záver

V tejto práci sme sa zoznámili so signovanými grafmi tak, že sme si najprv zadefinovali jednoduché grafy a potom sme ich rozšírili na signované grafy pridaním znamienok na hrany. Pre signované grafy sme potom definovali prepínanie vrcholov a balansovanosť kružníc. Ďalej sme riešili otázku, akou dátovou štruktúrou možno výhodne uchovať signované grafy v počítači a dospeli sme k riešeniu - dátového typu *igraph_t* knižnice *iGraph*.

V tretej kapitole sme definovali izomorfizmus jednoduchých grafov a izomorfizmus signovaných grafov. Povedali sme, že izomorfizmus signovaných grafov je závislý na izomorfizme jednoduchých (podkladových) grafov a jeden izomorfizmus podkladových grafov nestačí. K nájdeniu izomorfizmu podkladových grafov sme použili knižnicu *iGraph* a pridali do nej svoj modul s funkciami pre signované grafy. Hlavná funkcia implementovaného modulu rozhoduje, či sú dva zadané signované grafy izomorfné.

Popísali sme rozhrania funkcií implementovaného modulu, vstupy, výstupy jednotlivých funkcií a činnosť hlavných implementovaných algoritmov. Ku každému dôležitému algoritmu sme odhadli časovú zložitosť. Nakoniec sme k hlavnému programu, ktorý využíva funkcie z implementovaného modulu, vytvorili grafickú aplikáciu pre signované grafy. Môže slúžiť ako demonštračná aplikácia, v ktorej si používateľ graficky zadá signované grafy a potom si ich napríklad poprepína.

Ciele práce sme splnili. Našu *GUI* aplikáciu by ešte bolo dobré rozšíriť o ďalšie funkcionality a možnosti. Do modulu pre signované grafy by bolo možné pridať ďalšie algoritmy ako napríklad algoritmus, ktorý zistí, či je signovaný graf vyvážený respektive či je každá kružnica balansovaná. Okrem pridávania nových algoritmov by bolo možné vylepšiť zložitosť navrhnutých algoritmov, napríklad efektívnejším hľadaním bázových cyklov.

Literatúra

[1] R. Diestel: *Graph Theory, 4th edition*, Springer (2010)

[2] O. Holotňák: *Zbierka úloh z teórie grafov, diplomová práca*, Univerzita Komenského (2006)

[3] S. Palúch: *Algoritmická teória grafov, prvé vydanie*, Žilinská univerzita (2008)

[4] E. Rollová: Nowhere-zero flows and circuit covers of graphs and signed graphs, *dizertačná práca*, Univerzita Komenského (2012)

[5] R. Naserasr, E.Rollová, E.Sopena: *Homomorphisms of signed graphs*, Univ. Paris-Sud 11 France, Univ. Bordeauz France, Univerzita Komenského (2012)

[6] P. Hell, J.Nešetril: *Graphs and Homomorphisms*, Oxford Lecture Series in Mathematics and its Applications - Zväzok 28 (2004)

[7] F. Harary, *On the notion of balance of a signed graph*, Michigan Math Journal, Vol.2, pp.143-146 (1953)

[8] D. Cartwright, F. Harary: *Structural balance: a generalisation of Heider's theory*, Psychological Review 63, pp.277-293 (1956)

[9] Ladner, Richard E.:*On the structure of polynomial time reducibility*, Journal of the ACM 22, pp.151-171 (1975)

[10] T. Zaslavsky, *The geometry of root systems and signed graphs*, The American Mathematical Monthly, Vol.88, No.2, pp.88–105 (1981)

[11] T. Zaslavsky, *Characterizations of signed graphs*, Journal of Graph Theory, Vol.5, pp.401–406 (1981)

[12] T. Zaslavsky, *Signed graphs*, Discrete Applied Mathematics, Vol.4, pp.47–74 (1982)

[13] T. Zaslavsky, *A mathematical bibliography of signed and gain graphs and allied areas, 9.th edition*, Binghampton, New York, U.S.A., (1998)

[14] J.D. Ullman: *An Algorithm for Subgraph Isomorphism*, Journal of the ACM 23, pp.31-42 (1976)

[15] S. Skiena: *Graph Isomorphism*, In implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica. Reading, MA: Addison-Wesley, pp.181-187 (1990)

[16] The iGraph library - complex network research, `http://igraph.sourceforge.net/`

[17] G.Csárdi, T.Nepusz: *igraph Reference Manual*, `http://igraph.sourceforge.net/`

[18] Nauty and Traces - graph canonical labeling and automorphism group computation, `http://pallini.di.uniroma1.it/`

[19] B.D.McKay, A.Piperno: *Nauty and Traces User's Guide (Version 2.5)*, `http://www.http://pallini.di.uniroma1.it/Guide.html/`

[20] Bliss - a tool for computing automorphism groups and canonical labelings of graphs, `http://www.tcs.hut.fi/Software/bliss/`

[21] MinGW - Minimalist GNU for Windows, `http://www.mingw.org/`