UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Řešení Sudoku za pomoci teorie grafů

Bc. Kamil Štědrý

Diplomová práce
2010

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2009/2010**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Kamil ŠTĚDRÝ**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Řešení sudoku za pomoci teorie grafů**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

- V teoretické části práce bude popsána krátce hra sudoku, možnosti reprezentace problému řešení této hry za pomoci nástrojů teorie grafů a známé algoritmy, které lze k jejímu řešení využít. - Cílem práce je vytvořit/rozšířit program umožňující řešení hry sudoku algoritmy teorie grafů a grafickou reprezentaci principu řešení.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. CORMEN, H. A KOL. Introduction to algorithms. Boston, MIT Press, 2001. 2. LEWIS, H. R., DENENBERG, L. Data structures and their algorithms. Berkley, Adison-Wesley, 1997. 3. GOODRICH, M.T., TAMASSIA, R. Algorithm Design. Hoboken (NJ), John Wiley & Sons, 2002.

Vedoucí diplomové práce: **Ing. Tomáš Fidler**
Katedra softwarových technologií

Datum zadání diplomové práce: **30. října 2009**

Termín odevzdání diplomové práce: **21. května 2010**

prof. Ing. Simeon Karamazov, Dr.
děkan

L.S.

doc. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 10. listopadu 2009

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Chrudimi dne 24. 4. 2010 Bc. Kamil Štědrý

ANOTACE

Diplomová práce popisuje hru Sudoku a metody jejího řešení. Zaměřuje se především na možnost využití teorie grafů k jejímu řešení. V praktické části jsou implementovány vybrané metody a provedeno jejich porovnání.

KLÍČOVÁ SLOVA

Sudoku; teorie grafů; barvení grafu; řešení hrubou silou; algoritmus X

TITLE

Solving Sudoku with the help of graph theory

ANNOTATION

This thesis describes the Sudoku game and methods of its solution. It focuses primarily on the possibility of using graph theory for its solution. In the practical part are implemented some methods and carried out the comparison.

KEYWORDS

Sudoku; graph theory; graph coloring; brute force search; algorithm X

Obsah

Seznam zkratek

BFS	Breadth-first search (prohledávání do šířky)
CSV	Comma-separated values (souborový formát oddělující hodnoty čárkou)
DFS	Depth-first search (prohledávání do hloubky)
DLX	Dancing links (technika implementace algoritmu X)
FIFO	First in, first out (datová struktura – fronta)
LIFO	Last in, first out (datová struktura – zásobník)
NP	Nondeterministic polynomial (nedeterministicky polynomiální)
OOP	Object-oriented programming (objektově orientované programování)

Seznam obrázků

Seznam tabulek

Úvod

Cílem diplomové práce je stručné seznámení s hrou Sudoku a jejími pravidly. Dále budou popsány některé metody, které se k řešení Sudoku používají. Protože se práce zabývá řešením Sudoku za pomoci teorie grafů, je nezbytné uvést některé důležité pojmy z této oblasti. Z prostudovaných technik budou následně vybrány ty, které jsou vhodné právě pro řešení daného problému.

V praktické části práce pak bude vytvořen program využívající několika různých technik k řešení Sudoku umožňující grafickou reprezentaci hry. Program bude dále obsahovat generátor náhodných zadání Sudoku. Na základě těchto náhodných zadání bude provedeno porovnání implementovaných metod a zhodnocení vhodnosti využití znalosti teorie grafů k řešení Sudoku.

1 HRA SUDOKU

V úvodu je nutné popsat samotnou hru Sudoku a její pravidla. Důležité je také vymezení používaných pojmů, aby nedocházelo k neporozumění. Zajímavá je také historie této hry nebo její různé varianty.

1.1 Historie

Hra Sudoku bývá často nesprávně označována za japonský číselný hlavolam, její historie je ovšem složitější. Kořeny hry musíme hledat již ve starověké Číně kolem roku 2200 př. n. l. ve hře *Lo Shu*, která vznikla na základě čínské legendy. Podle legendy zem tehdy zachvátila potopa. Přívaly vody neustávaly, i když lidé neustále vykonávali obětní rituály. Potom si jedno děvče všimlo, že se z řeky *Lo* vynořila želva, která měla na svém krunýři tečky znázorňující číslice 1 až 9. Byly uspořádány do třech řad a třech sloupců tak, že součet čísel vodorovně, svisle i šikmo byl vždy patnáct. Lidé pochopili, že musí vykonat patnáct obětí, tak jak jim ukázaly číslice. Když tak učinili, voda začala ustupovat (1). Hra *Lo Shu* se stala později známou i ve zbytku světa pod názvem Kouzelné čtverce (Magic square). Na základě této hry vymyslel v 18. století švýcarský matematik Leonhard Euler hru Latinské čtverce (Latin square). Což je tabulka o $n \times n$ polích, která je vyplněna n různými symboly tak, že v každém řádku i v každém sloupci se každý symbol nachází právě jednou.

Odtud je již jen krůček k samotnému Sudoku. Číselnou hádanku pod názvem *Number Place* (Umísti číslice) vymyslel v roce 1979 *Howard Garns*, 74letý architekt a tvůrce rébusů z Indiany. V roce 1984 byla uvedena v Japonsku pod názvem *Suuji wa dokushin ni kagiru*, což může být přeloženo jako „Čísla musí být samotná“. Hra se stala v Japonsku velice populární, zejména proto, že zde prakticky neexistují klasické křížovky. Protože byl název příliš dlouhý, začalo se hře říkat *Sú doku (sú* – číslice, *doku* – osamocený). Od roku 2005 se stal rébus celosvětovým hitem a začal se pro něj používat anglický přepis japonského názvu Sudoku. To je tedy důvodem, proč je Sudoku mnohými považováno za původem japonskou hru.

1.2 Princip hry

Obecně lze Sudoku hádankou nazvat pole o rozměrech $n^2 \times n^2$ políček logicky dělených na bloky $n \times n$ políček. Nejčastějším typem je hádanka, kde $n = 3$ a znázorňuje tedy hrací pole o devíti řádcích a devíti sloupcích, rozdělených na devět bloků o velikosti 3×3 políček (2). Vyplněním některých číslic do hracího pole získáme zadání Sudoku, neboli Sudoku úlohu, kterou můžeme vidět na obrázku 1. Počet nevyplněných hodnot ovšem nevypovídá o složitosti dané hádanky. Složitost je dána vzájemnými vazbami mezi políčky, které nejsou na první pohled patrné.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Obrázek 1 – Sudoku úloha (2)

Předpokladem pro správné vyřešení rébusu je jednoznačnost úlohy. Jednoznačnost znamená, že není možné doplnit číslice jiným způsobem tak, aby platila pravidla Sudoku. Cílem hry je pak do hracího pole doplnit chybějící číslice 1 až 9 tak, aby platilo, že:

- v každém řádku se musí každá číslice vyskytovat právě jednou,
- v každém sloupci se musí každá číslice vyskytovat právě jednou,
- v každém bloku 3×3 políček se musí každá číslice vyskytovat právě jednou.

1.3 Vymezení pojmů

Aby nedošlo při dalším vysvětlování k nedorozumění, zde jsou uvedeny některé základní pojmy spojené s hrou Sudoku:

- *Pole Sudoku* (hrací pole) je obecně hrací pole Sudoku bez rozlišení, je-li vyplněné či ne.
- *Řešení Sudoku* (úplné pole Sudoku) je takové pole Sudoku, které je zcela vyplněno dle pravidel Sudoku.
- *Zadání Sudoku* (Sudoku úloha) označuje pole Sudoku obsahující nevyplněná políčka.
- *Políčko* (buňka) je jedno políčko, které může obsahovat hodnotu. Pole Sudoku má právě 81 políček.
- *Hodnota* (číslice) je celé číslo z intervalu 1 až 9, které může být vyplněno v políčku.
- *Blok* (box) značí čtverec o rozměrech 3 × 3 políčka. Pole Sudoku je logicky členěno na devět takových bloků číslovaných zleva doprava, od vrchu dolů čísly 1 až 9.
- *Kandidát* je celé číslo z intervalu 1 až 9. Značí číslici, která může být doplněna do daného políčka. Ke každému políčku může existovat více kandidátů.
- *Sousední buňky* (peers) jsou buňky nacházející se ve stejném řádku, sloupci nebo bloku jako vybraná buňka. Sousední buňky tedy nemohou obsahovat stejnou hodnotu jako vybraná buňka.
- *Skupina* označuje celý řádek, sloupec nebo blok.
- *Vodítko* (clue) je hodnota vyplněná již v zadání.

1.4 Omezení hry

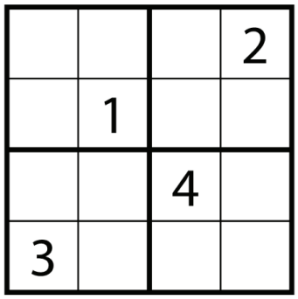
Jedním z náročných a zajímavých problémů hry Sudoku je zodpovězení otázky: „Kolik celkem existuje zadání Sudoku?“ Touto otázkou se důkladně zabýval matematik *Bertram Felgenhauer*, který za pomoci specifického algoritmu vypočetl existenci 6 670 903 752 021 072 936 960 unikátních zadání Sudoku (3). Toto číslo se stalo respektovaným odbornou veřejností z celého světa.

Další zajímavou otázkou ohledně zadání Sudoku je: „Jaký je nejnižší počet vodítek v zadání Sudoku?“ Tato otázka ještě není s určitostí zodpovězena. V současné době je známo více než 36 000 zadání Sudoku se sedmnácti vodítky, ale žádné s šestnácti nebo menším počtem vodítek. Za minimální počet vodítek zadání Sudoku se tedy považuje sedmnáct. Pro symetrické Sudoku je minimem osmnáct vodítek (3).

Třetí otázka zní: „Jaký je minimální počet různých hodnot vodítek v zadání Sudoku?“ Na rozdíl od předchozích otázek je odpověď na tuto otázku s určitostí přesná. U klasického Sudoku je to osm různých hodnot. Obecně lze říci, že minimální počet různých hodnot vodítek je $n^2 - 1$, jinak není zadání jednoznačné (4).

1.5 Varianty

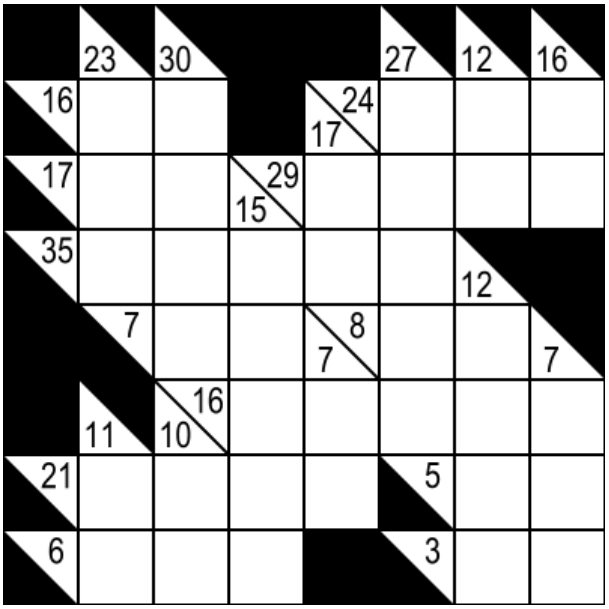
Nejrozšířenější je bezesporu Sudoku o devíti řádcích a devíti sloupcích rozdělených do bloků 3 × 3 políček, kde je úkolem doplnit číslice 1 až 9. Kromě této varianty existuje celá řada podobných hádanek. Jak již bylo řečeno, Sudoku může být každé pole o rozměrech $n^2 \times n^2$ políček. Pro $n = 2$ vznikne pole o čtyřech řádcích a čtyřech sloupcích rozdělené na bloky 2 × 2 políček. Toto Sudoku bývá někdy nazýváno *Shidoku* a do políček se doplňují číslice 1 až 4. Na obrázku 2 je zobrazeno *Shidoku* s minimálním možným počtem vodítek. Obdobně pro $n = 4$ získáme pole 16 × 16 políček rozdělené na bloky 4 × 4 políček.



Obrázek 2 – Shidoku (3)

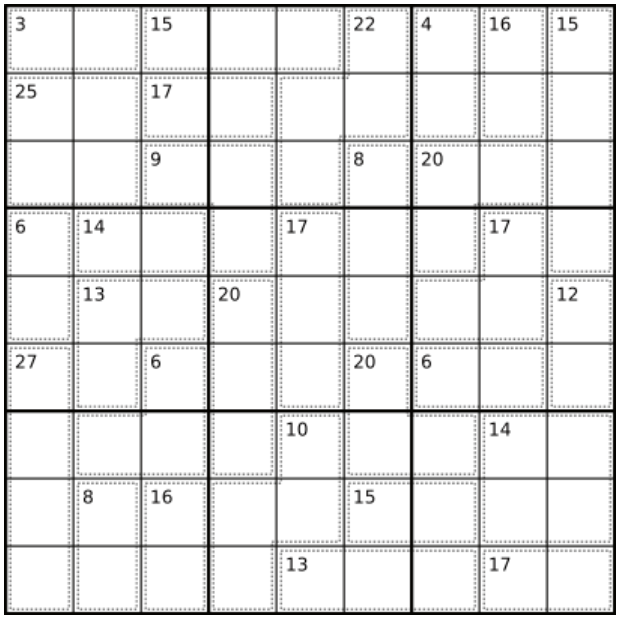
Použití číslic v Sudoku je nepochybně nejuniverzálnější pro použití po celém světě. Přesto existují hlavolamy, ve kterých se místo čísel používají například písmena. V dalších variantách se doplňují různé barvy, obrázky, případně znaky či symboly.

Mezi vzdálenější příbuzné Sudoku patří *Kakuro*, ve kterém je cílem doplněním číslic dosáhnout určitého součtu v řádcích a sloupcích (5). Jednoduché zadání hry Kakuro je zobrazeno na obrázku 3.



Obrázek 3 – Kakuro(5)

Z něho nepochybně vychází další varianta, *Killer Sudoku*, ve které kromě základních pravidel Sudoku platí ještě dosažení určitého součtu skupin po několika políčkách (6). Jedna z možných reprezentací je znázorněna na obrázku 4.



Obrázek 4 – Killer Sudoku (6)

Poslední zajímavou mechanickou obdobou je *Sudoku Cube*, pro kterou byl základem známý hlavolam *Rubikova kostka*. Má tedy každou stěnu rozdělenou na devět políček (3 × 3) a jednotlivé řady nebo sloupce je možné otáčet kolem středových os. Úkolem pak je otáčením naskládat na každou stěnu číslice 1 až 9. Obrázek 5 znázorňuje složenou *Sudoku Cube* (7).

Obrázek 5 – Sudoku Cube (7)

2 METODY ŘEŠENÍ ÚLOH SUDOKU

Pro řešení úloh Sudoku lze použít celou řadu metod. V zásadě je lze rozdělit do kategorií:

- metody založené na způsobu řešení člověkem,
- metody systematického procházení.

Za další kategorii je možné označit genetické metody řešení založené na teorii grafů. Ověření vhodnosti tohoto přístupu bude cílem dalšího textu práce.

2.1 Metody založené na způsobu řešení člověkem

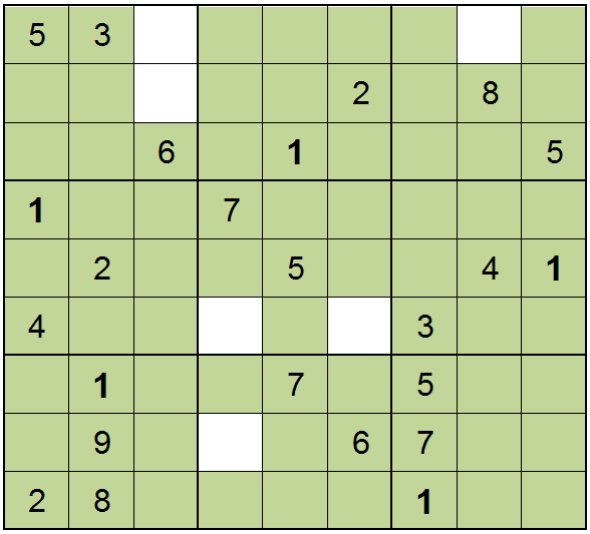
Tyto metody vycházejí z postupu lidského řešitele. Mohou sloužit přímo k přiřazení čísla do daného políčka, nebo alespoň k omezení počtu kandidátů tohoto políčka. Použití těchto metod vede k velmi rychlým algoritmům. Pro vyřešení všech zadání Sudoku je ovšem nutností použít kombinace několika takových metod, přičemž se celý algoritmus zpomaluje a komplikuje se i jeho zápis. Při ideální aplikaci jednotlivých metod je zaručeno vyřešení všech zadání, která dokáže vyřešit člověk. Pro popis následujících metod jsem použil zdroj (8), který tuto problematiku uvádí komplexně a zároveň přehledně.

2.1.1 Jediná možnost

Jednoduchou metodou, která umožňuje určit přímo číslo, jež má být v políčku zapsáno, je metoda „jediná možnost“. Jejím smyslem je zjištění všech kandidátů daného políčka. Pokud je jediný kandidát, je možné ho zapsat do políčka. Znamená to tedy, že sousední buňky obsahují již zbylých osm číslic. Pokud políčko nemá žádného kandidáta, znamená to, že řešení obsahuje chybu.

2.1.2 Jediné ve skupině

Tato metoda vychází ze základního pravidla Sudoku, které říká, že ve skupině se může každé číslo vyskytovat právě jednou. Pokud tedy zjistíme, že ve skupině existuje jediné pole, na které můžeme zapsat číslo při dodržení tohoto pravidla, máme vyhráno.



Obrázek 6 – Jediné ve skupině

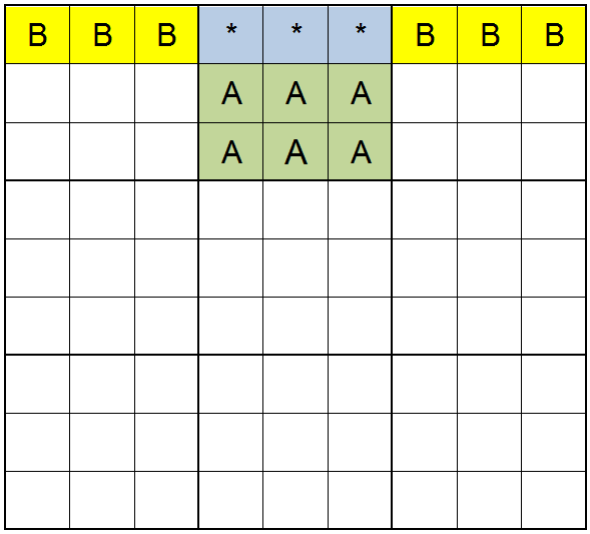
Na obrázku 6 je zobrazena situace, kdy chceme doplnit chybějící jedničky. Jedničky mohou být doplněny pouze na bílá políčka, protože ostatní sousedí s políčkem, v němž již jednička vyplněna je. Je tedy zřejmé, že jedničku lze v druhém řádku doplnit pouze na třetí pozici. Ve třetím bloku je také jediná možnost doplnění jedničky, a to v prvním řádku a osmém sloupci. Obdobně lze doplnit i zbylé jedničky.

2.1.3 Holé množiny

Jde o metodu omezující počet kandidátů. Podstatou metody je situace, kde se v jedné skupině vyskytuje *n* políček, která obsahují právě *n* kandidátů. Nejjednodušší varianta znamená dvě políčka a dva kandidáty. Je tedy jasné, že ve výsledku se tato dvě políčka podělí o dvě vyhovující čísla. Tím pádem je možné vymazat tato dvě čísla ze seznamu kandidátů ostatních políček skupiny. Zcela obdobná situace platí samozřejmě i pro tři čísla na třech políčkách až po sedm čísel na sedmi políčkách.

2.1.4 Průsečíky skupin

Jedná se vždy o průsečík bloku s řádkem nebo sloupcem. Průsečíkem mohou být tedy maximálně tři políčka. Na obrázku 7 jsou tato políčka označena hvězdičkou.



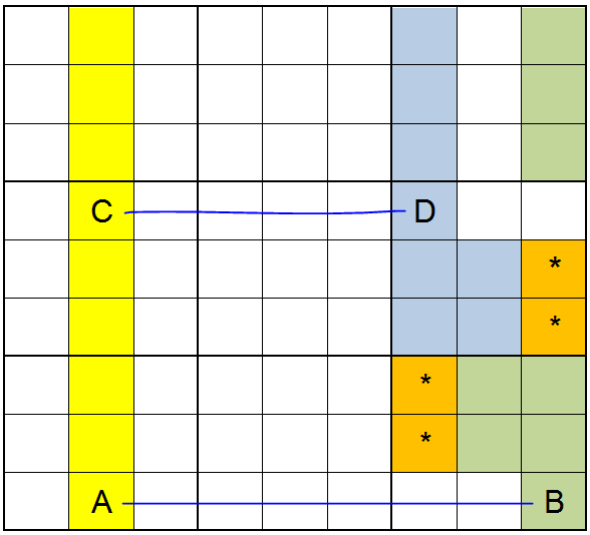
Obrázek 7 – Průsečík skupin

Pokud se v políčkách průsečíku vyskytují dvě nebo tři stejné číslice jako kandidáti, mohou nastat dvě významné situace:

- Pokud se číslo z průsečíku již nikde jinde v řádku nevyskytuje a vyskytuje se pouze na některých ostatních buňkách bloku označených *A*, plyne z toho jednak to, že nutně musí být číslice na některé buňce průsečíku a jednak fakt, že se tedy nemůže vyskytnout v žádné buňce označené *A*, a proto je možné ji z těchto buněk vyškrtnout.
- Druhá varianta nastává v případě, že se kromě průsečíku vyskytuje číslo na některém z polí označených *B* a nevyskytuje se ve zbylé části bloku. Znamená to, že musí být číslice na některém z polí průsečíku a nemůže tedy být na žádném poli označeném *B*. Opět lze hodnotu vymazat ze seznamu kandidátů polí označených *B*.

2.1.5 Dvojice

Opět se jedná o metodu eliminující počet kandidátů. Tuto metodu je možné použít, pokud existují dvě silně vázané dvojice (dvojice čísel, které se vyskytují jako kandidáti samy ve skupině). Jeden konec každé z dvojic končí ve stejné skupině.

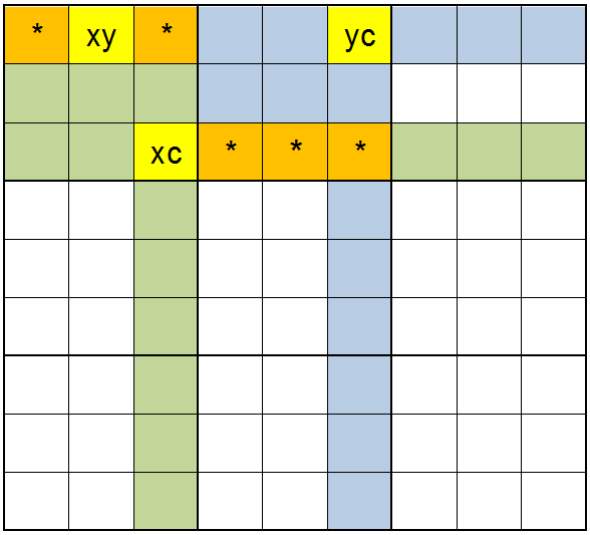


Obrázek 8 – Dvojice

Na obrázku 8 je znázorněn příklad, kdy jsou dány vázané dvojice *AB* a *CD*, přičemž políčka *A* a *C* jsou ve stejném sloupci. Jsou znázorněny sousední buňky druhých konců dvojic a jejich průsečíky označeny hvězdičkou. Nyní je jasné, že se čísla z vázaných dvojic nemohou vyskytnout v políčkách označených hvězdičkou, proto je můžeme odstranit ze seznamu kandidátů.

2.1.6 XY křížení

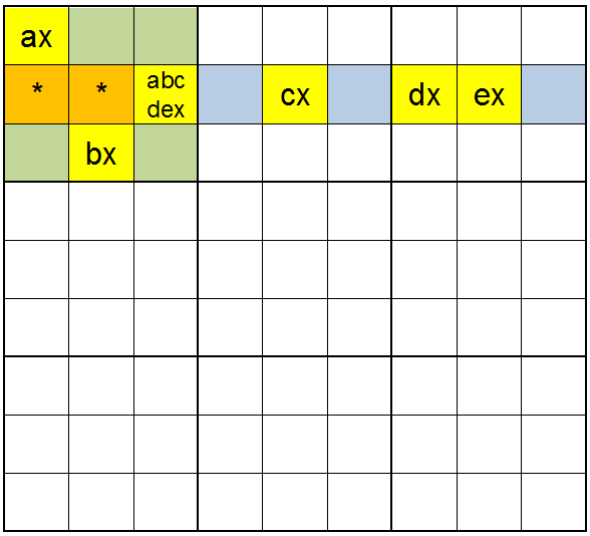
Situace při této metodě vždy vychází z vedoucího políčka, na kterém existují dva kandidáti. Vedoucí políčko musí mít mezi sousedními buňkami další dvě, které obsahují opět dvojici kandidátů, přičemž jeden z kandidátů je totožný s jedním z kandidátů na vedoucím políčku a s druhým kandidátem pro obě políčka společným. Na obrázku 9 je znázorněna situace, kdy vedoucí políčko obsahuje kandidáty *x* a *y*. Jeho dvě sousední políčka obsahují kandidáty *xc* a *yc*. Průsečík sousedních buněk těchto políček je označen hvězdičkou. Je zřejmé, že políčka označená hvězdičkou nemohou obsahovat číslici *c* a je možné ji vymazat ze seznamu kandidátů.



Obrázek 9 – XY křížení

2.1.7 Vedoucí prvek

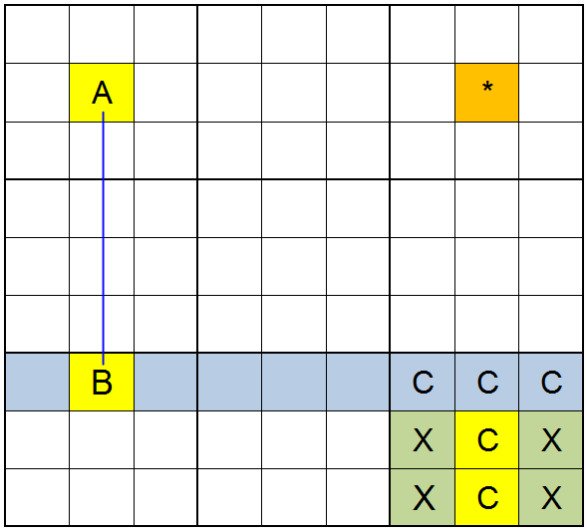
U této metody se vždy jedná o dvojici blok – řádek nebo blok – sloupec. Oproti metodě XY-křížení můžeme mít ve vedoucím prvku navíc další kandidáty, včetně kandidáta na odstranění. To samozřejmě vyžaduje, aby se na políčkách, která s vedoucím prvkem sousedí, vyskytly všechny kombinace kandidáta na zrušení se všemi ostatními kandidáty na vedoucím poli. Tím se omezí i množina prvků, na kterých můžeme vyškrtnout kandidáta *x*, protože s těmito políčky musí všechna dotyčná políčka sousedit. Na obrázku 10 to tedy znamená, že společnou podmnožinou, sousedící se všemi zúčastněnými políčky, jsou políčka označená hvězdičkou.



Obrázek 10 – Vedoucí prvek

2.1.8 Prázdný čtverec

Obrázek 11 zobrazuje ve druhém sloupci silně vázanou dvojici nějakého kandidáta *n*. V bloku číslo 9 jsou na políčkách označených *X* buď vyřešená čísla, nebo jsou to pole, kde *n* není kandidátem. Naopak na políčkách označených *C* se číslo *n* jako kandidát může vyskytovat, a musí se vyskytovat alespoň na jednom ze světlých políček se symbolem *C*. Pokud se v takovémto schématu vyskytne na políčku označeném hvězdičkou kandidát *n*, můžeme jej vyškrtnout.



Obrázek 11 – Prázdný čtverec

2.2 Metody systematického procházení

Tyto metody procházejí rekurzivně celou oblast řešení a hledají taková řešení, která vyhovují daným podmínkám. To umožňuje nalezení více možných řešení, ale zároveň znamená prodloužení doby hledání. Při použití algoritmů pro řešení Sudoku úloh vyžaduje vždy ověření existence více možných řešení. Toho je dosaženo dalším prohledáváním v oblasti řešení, dokud není nalezeno další řešení nebo není vyvrácena možnost existence dalších řešení.

2.2.1 Řešení hrubou silou

Řešení hrubou silou (brute force search) je způsob řešení úlohy, při kterém se systematicky prochází celý prostor možných řešení problému. Jeho výhodou je nalezení nejlepšího možného řešení nebo případný důkaz o nemožnosti řešení problému. Nevýhodou bývá velká složitost hledání, tedy časová a případně paměťová náročnost algoritmu (9).

Při řešení úlohy Sudoku se tedy postupuje zleva doprava, shora dolů a zkouší se dosadit číslo jedna. Pokud přestanou platit pravidla Sudoku, zkouší se vždy číslo o jedno vyšší. Výsledná složitost pro klasickou Sudoku úlohu je 9^{81}. Experimentální zjištění ukazují, že při dobře navrženém algoritmu je použití metody hrubé síly dobrým způsobem řešení Sudoku. Metoda buď najde řešení, nebo zjistí, že zadání nemá řešení. Jednoduchým rozšířením lze testovat i možnost více řešení. Ve srovnání s logickým postupem řešení může být značně pomalejší. Neplatí ovšem vztah mezi složitostí zadání Sudoku a dobou řešení metodou hrubé síly. I docela jednoduché zadání může být touto metodou řešeno dlouhou dobu. Záleží na rozmístění vodítek a prázdných buněk (10). Příkladem může být zadání na obrázku 12.

					3		8	5
		1		2				
			5		7			
		4				1		
	9							
5							7	3
		2		1				
				4				9

Obrázek 12 – Těžké zadání pro metodu hrubé síly (10)

Algoritmus prochází hrací pole zleva doprava, shora dolů. Jelikož je celý první řádek prázdný, musí projít velké množství možností, než určí správně devítku v levém horním rohu a následně další číslice. Řešení tohoto zadání vyžaduje 641 580 843 pokusů o doplnění číslice a na počítači s 3GHz procesorem zabere 30 až 45 minut času. Algoritmus lze dále vylepšit, například před samotným řešením celé Sudoku pole otočit o násobek 90° a získat tak lepší počáteční pozici. Volbu programovacího jazyka je nutné také pova-

žovat za důležitou. Při volbě jazyka s nízkou režií (obecně neobjektové jazyky) lze dosahovat rychlejšího řešení. S těmito modifikacemi trvá pak vyřešení uvedeného zadání přibližně půl sekundy. Tohoto zrychlení je dosaženo již jednou uvedeným otočením původního Sudoku pole, čímž se do vrchního řádku dostanou některé číslice ze zadání a rapidně se sníží množství zpětného procházení.

Řešení hrubou silou je možné dále rozdělit podle způsobu prohledávání stavového prostoru (11):

- BFS (breadth-first search – prohledávání do šířky) postupně prochází všechny vrcholy v daném stavovém prostoru. Algoritmus nejprve projde všechny sousedy startovního vrcholu, poté sousedy sousedů atd. až projde celý stavový prostor. Následníci startovního prvku a dále následníci prohledávaných prvků jsou vkládány do FIFO fronty, odkud jsou brány k prohledávání ve stejném pořadí, jako byly vloženy.
- DFS (depth-first search – prohledávání do hloubky) postupně prochází všechny vrcholy v daném stavovém prostoru. Používá backtracking (zpětné vyhledávání), které zkouší vyhledávat řešení metodou „pokus – omyl“. Následníci startovního prvku a dále následníci prohledávaných prvků jsou ukládány do LIFO zásobníku, odkud jsou brány k prohledávání v opačném pořadí, než byly vloženy.

2.2.2 Algoritmus X

Autorem tohoto algoritmu je počítačový vědec *Donald Ervin Knuth*. Jedná se o rekurzivní, nedeterministický algoritmus, prohledávající do hloubky a využívající backtracking. Algoritmus hledá všechna řešení, která splňují podmínky přesného pokrytí (exact cover) zadaného problému, jenž je reprezentován maticí skládající se z jedniček a nul.

Předpokládejme, že máme dánu množinu S složenou z podmnožin hledané kombinace stavů X. Přesné pokrytí je taková podmnožina S^* z množiny S, která obsahuje prvky z X právě jednou. Dá se říci, že každý element v X je pokryt právě jednou v množině S^*.

V matici jsou zakódovány všechny stavy prohledávaného prostoru. Každý řádek obsahuje část řešení a algoritmus hledá ty řádky, které neobsahují duplicitní informace

o řešení. Vybrané řádky nesmí mít jedničky ve stejných sloupcích a zároveň musí mít dohromady jedničky ve všech sloupcích.

Obecný postup algoritmu vypadá pak následovně:

1. Pokud je matice *A* prázdná, problém je vyřešen a algoritmus úspěšně končí.
2. Vyber sloupec *c* (deterministicky).
3. Vyber řádek *r*, ve kterém platí $A_{r,c} = 1$ (nedeterministicky).
4. Vlož řádek *r* do částečného řešení.
5. Pro každý sloupec *j*, ve kterém platí $A_{r,j} = 1$, a pro každý řádek *i*, ve kterém platí $A_{i,j} = 1$ vymaž řádek *i* z matice *A*, a vymaž sloupec *j* z matice *A*.
6. Opakuj algoritmus na zredukované matici od kroku 1.

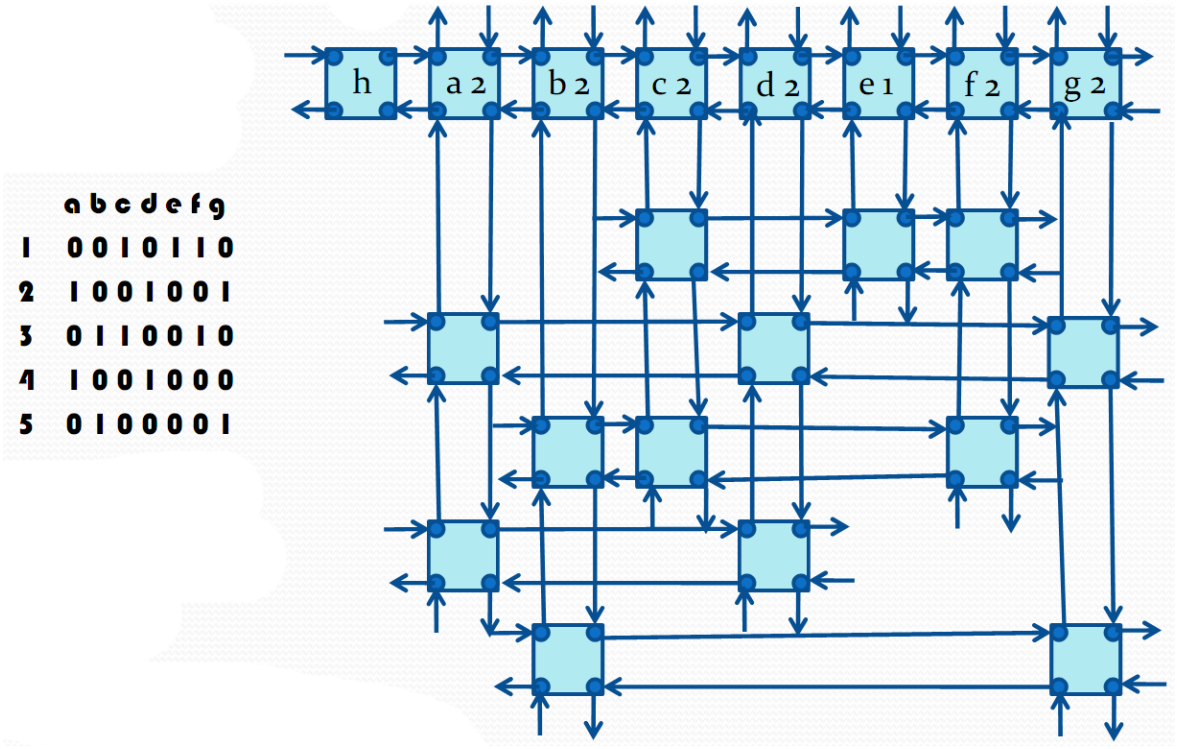
Podstatou algoritmu je vytvoření matice s omezujícími podmínkami pro každé políčko hry Sudoku a každou možnou číslici. Což u klasického Sudoku představuje matice o 729 řádcích ($9 \times 9 \times 9$). Ve sloupcích matice jsou omezení pro řádek, sloupec, blok a konkrétní číslici. Dále ještě omezení určující políčku právě jednu hodnotu. To ve výsledku dá 324 sloupců v matici omezení. Matice je pak ohodnocena čísly 0 a 1, které značí, zda se kandidát může vyskytovat na dané pozici. Sudoku je úspěšně vyřešeno, pokud se v každém sloupci vyskytuje jedna hodnota 1. Implementace algoritmu X se často realizuje pomocí techniky DLX.

2.2.3 Dancing links

Technika Dancing links, obecně známá pod zkratkou DLX, byla vyvinuta *Donaldem Knuthem* k efektivní implementaci Algoritmu X na počítači. DLX používá matici, která je tvořena cyklickými obousměrnými spojovými seznamy. Pro každý řádek a sloupec existuje v matici jeden seznam. Každá buňka s jedničkou obsahuje odkaz na své sousedy nahoře, dole, vlevo, vpravo a také uchovává informaci, ve kterém sloupci se nachází.

Hlavní myšlenka této implementace plyne z pozorování vlastností cyklických obousměrných seznamů. V této struktuře lze jednoduše přidávat a odebírat prvky při zachování propojení. Tato technika funguje nezávisle na počtu prvků a pracuje stejně i v případě jed-

noho prvku v seznamu. Příklad reprezentace matice pomocí techniky DLX je znázorněn na obrázku 13.



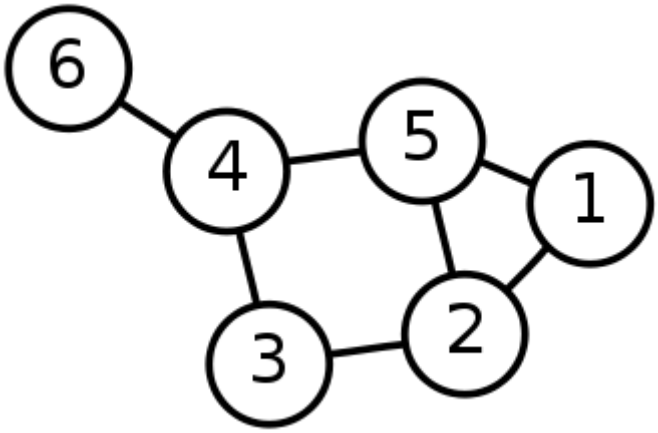
Obrázek 13 – Příklad reprezentace matice pomocí DLX (11)

3 TEORIE GRAFŮ

V následující části textu budou vysvětleny základní pojmy teorie grafů, které je nutné znát k pochopení dále popsaných technik řešení. Cílem není komplexní popsání oblasti teorie grafů, ale pouze seznámení čtenáře s používanými termíny.

3.1 Základní pojmy

Nyní je vhodné vysvětlit, co grafy jsou. Jak uvádí Jarovský (12), nejedná se o grafy statistické nebo grafy funkcí. Graf si můžete představit jako strukturu, která zjednodušuje reálné objekty. Ty jsou znázorněny pomocí bodů a čar, které je spojují, a tím popisují daný problém. Grafy slouží jako abstrakce mnoha různých problémů. Často se jedná o zjednodušený model nějaké skutečné sítě (například dopravní). Zdůrazňuje topologické vlastnosti objektů, zejména jejich vzájemné propojení a zanedbává geometrické vlastnosti (například polohu). Na obrázku 14 je zobrazen jednoduchý graf.



Obrázek 14 – Jednoduchý graf (13)

Dále jsou uvedeny základní pojmy z teorie grafů, které jsou důležité pro aplikaci řešení Sudoku úloh:

- *Graf* je základním objektem teorie grafů. Je to uspořádaná dvojice (*V*, *E*), kde *V* je neprázdná množina a *E* je množina dvojic prvků z *V*.
- *Vrchol* (uzel) je prvek z množiny *V*. Vrcholy jsou body ve zjednodušené struktuře.
- *Hrana* je prvek z množiny *E*. Hrany spojují dvojice vrcholů.

- *Orientovaná hrana* je hrana spojující dvojici vrcholů pouze jedním směrem.
- *Smyčka* je hrana vedoucí z vrcholu do něj samotného.
- *Násobné hrany* spojují vícekrát stejné vrcholy.
- *Incidence* (sousednost) řekneme, že prvek *i* sousedí s prvkem *j*, pokud z *i* vede hrana do *j*.
- *Stupeň vrcholu* označuje počet hran, které do daného vrcholu zasahují.
- *Orientovaný graf* je graf, který obsahuje orientované hrany.
- *Úplný graf* označuje neorientovaný graf, v němž jsou každé dva vrcholy spojené hranou.
- *Rovinný (planární) graf* je graf, pro který existuje takové rovinné nakreslení, že se žádné dvě hrany nekříží.
- *Barevnost grafu* (chromatické číslo grafu) je nejmenší počet barev, který je potřebný k obarvení grafu.

3.2 Reprezentace grafů

Existuje několik různých způsobů jak reprezentovat (popsat) strukturu grafu. Graf je možné popsat:

- diagramem (nákresem),
- definicí,
- maticí,
- datovou strukturou (v paměti počítače).

Pro účely programování jsou důležité poslední dva způsoby, které budou dále popsány.

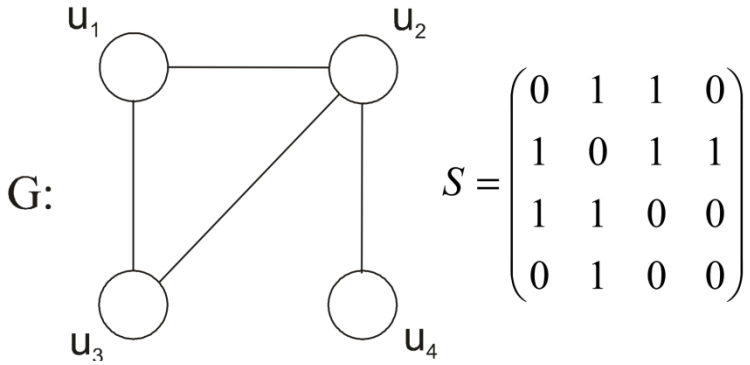
3.2.1 Maticový popis grafu

Maticový popis grafu vychází ze dvou základních vztahů v grafu. Prvním je vztah mezi hranou a jejím koncovým uzlem (vztah incidence). Ten popisuje matice incidence grafu. Druhým vztahem je sousednost uzlů. Ten popisuje matice sousednosti grafu.

Uvažujeme neorientovaný graf, který neobsahuje smyčky ani násobné hrany. Matice sousednosti je čtvercová matice řádu *m*, kterou označíme S. Její prvky S_{jk} jsou dány předpisem:

- $S_{jk} = 1$, jestliže uzly U_j a U_k jsou sousední,
- $S_{kj} = 0$, jestliže uzly U_j a U_k nejsou sousední.

Z této definice vyplývá, že je výsledná matice sousednosti symetrická podle hlavní diagonály, na hlavní diagonále má nuly a počet jedniček v matici je roven dvojnásobku počtu hran. Příklad grafu s maticí sousednosti je zobrazen na obrázku 15.



Obrázek 15 – Graf s maticí sousednosti (14)

3.2.2 Reprezentace datovou strukturou

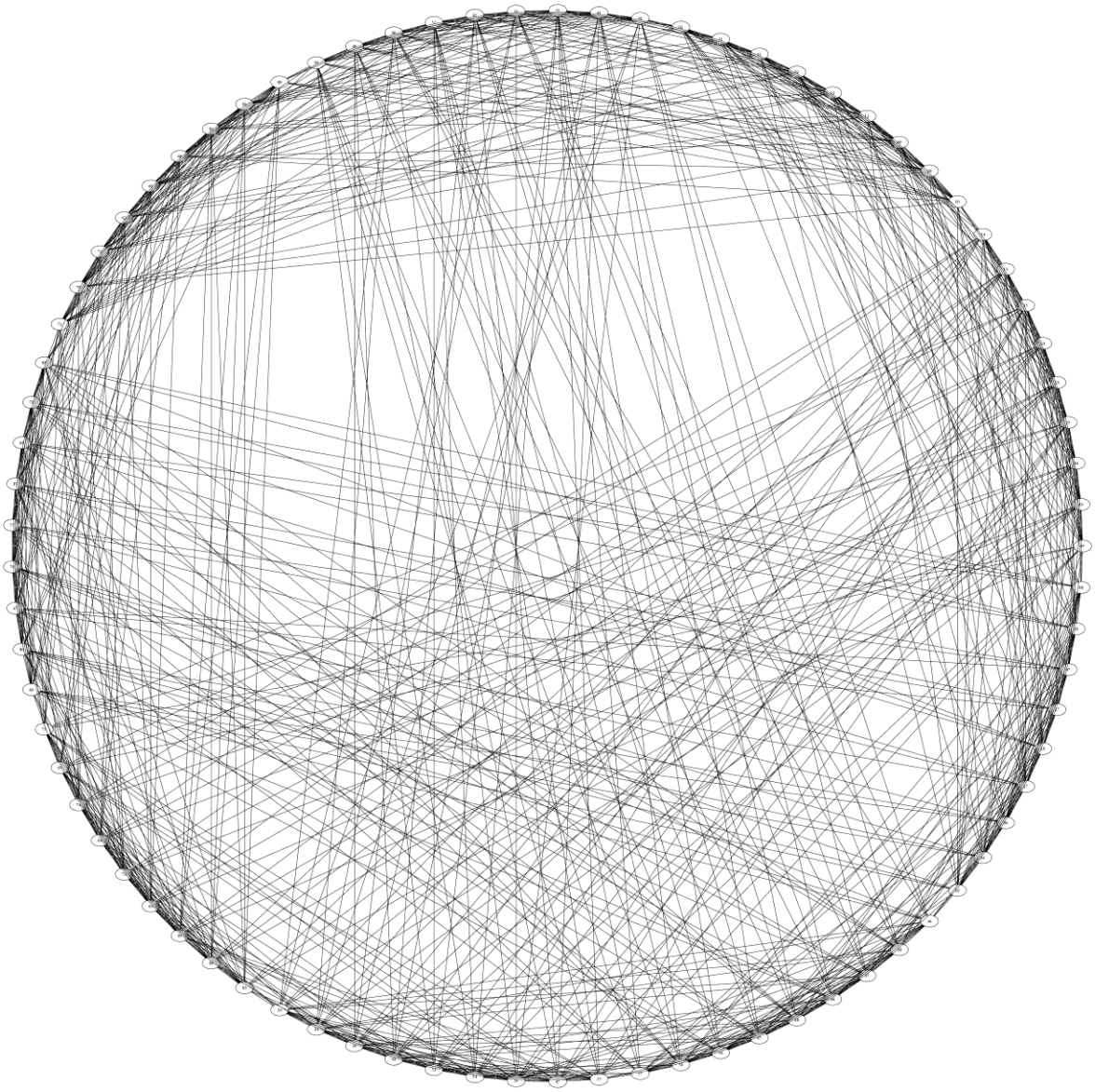
Graf můžeme v programu reprezentovat různými způsoby. Můžeme ho uložit i jako matici sousednosti. Tento způsob ovšem není zcela efektivní, zvláště při ukládání grafů s velkým počtem vrcholů. Za efektivní nelze pokládat ani procházení takto reprezentovaných grafů.

Pro zefektivnění můžeme graf reprezentovat pomocí dvou polí. První pole má stejný počet prvků, jako je počet vrcholů v grafu. Každému vrcholu odpovídá jeden prvek pole. V něm je uložena hodnota indexu, od kterého v druhém poli začíná seznam vrcholů, jež sousedí s tímto vrcholem.

Druhou možností je použití dynamické datové struktury. Každý vrchol je reprezentován jako datový typ, který obsahuje seznam ukazatelů na sousední vrcholy. Tento typ reprezentace je vhodné použít na grafy, u kterých se počet vrcholů nebo hran často mění v čase.

3.3 Reprezentace Sudoku

Pro aplikování teorie grafů k řešení Sudoku úloh je nutné pole Sudoku reprezentovat jako graf. Vrcholy představují jednotlivá políčka hry Sudoku a hrany znázorňují vazby mezi nimi. Dva vrcholy budou spojeny hranou, pokud jsou umístěny v původním poli Sudoku ve stejném řádku, sloupci nebo bloku. Což znamená, že každý vrchol bude spojen hranou s dvaceti jinými vrcholy. Získáme tedy graf o 81 vrcholech a 810 hranách, který je znázorněn na obrázku 16.



Obrázek 16 – Grafová reprezentace Sudoku

4 ŘEŠENÍ SUDOKU NA ZÁKLADĚ TEORIE GRAFŮ

V této kapitole budou popsány obecné postupy teorie grafů, které lze po vhodném přizpůsobení využít pro řešení Sudoku.

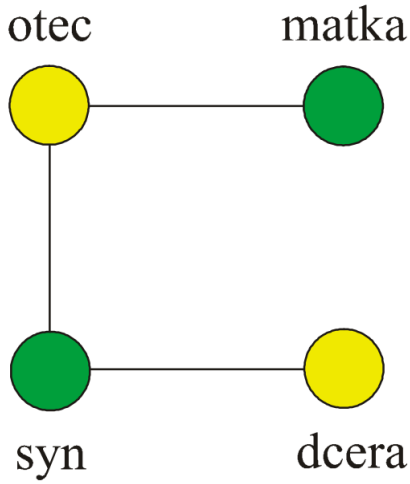
4.1 Barvení grafu

Barvení grafu je jednou z disciplín teorie grafů, která se zabývá přiřazováním barev objektům grafu tak, aby sousední objekty nebyly obarveny stejnou barvou. Princip barvení grafu je možno použít při řešení různých úloh ze skutečného světa. Příkladem může být problém přidělování zdrojů nebo problém barvení mapy.

Cílem úlohy přidělování zdrojů je nalezení nejmenšího počtu potřebných zdrojů, které sdílejí jednotlivé objekty. Princip barvení grafu je možno demonstrovat na následujícím příkladu, který je převzat od Večerky (14). Máme rodinu, jejíž příslušníci potřebují automobil v určitých hodinách dne, a to následovně:

- Otec odjíždí autem v sedm hodin do zaměstnání a domů se vrací v šestnáct hodin.
- Matka je doma a auto potřebuje dopoledne od devíti do dvanácti hodin na nákupy.
- Syn auto potřebuje odpoledne od patnácti hodin, kdy odjíždí hrát volejbal.
- Dcera auto potřebuje od osmnácti hodin, kdy jede za svým přítelem.

Následně sestavíme graf, ve kterém budou dva uzly sousední, když dva rodinní příslušníci potřebují automobil ve stejnou dobu. Následně graf obarvíme tak, aby žádné dva sousední vrcholy neměly stejnou barvu, a přitom jsme použili nejmenší možný počet barev. Pro obarvení takového grafu jsou potřeba dvě barvy, jak je vidět na obrázku 17.



Obrázek 17 – Jednoduchý obarvený graf (14)

Další častá úloha řeší problém barvení politické mapy. Jejím cílem je obarvení mapy států tak, aby žádné dva sousední státy neměly stejnou barvu. Takovou mapu můžeme reprezentovat jako graf, ve kterém vrcholy představují jednotlivé státy a hrany jejich sousednost. Jak je patrné, tento problém je již poněkud složitější. Teprve nedávno (v 70. letech 20. století) byl nalezen matematický důkaz, že k obarvení jakékoliv mapy států stačí čtyři barvy. Což lze tvrdit i obecně o všech rovinných grafech.

4.1.1 Obecný heuristický algoritmus barvení grafu

Jednotlivé barvy, které použijeme k barvení grafu, budeme značit přirozenými čísly (1, 2, …). Proměnná *B* bude označovat nejvyšší číslo použité barvy a zároveň počet použitých barev. Číslo barvy, kterou je obarven uzel *u*, budeme označovat funkcí *b*, tj. zápisem *b(u)*. Skutečnost, že uzel *u* obarvíme barvou *c*, zapíšeme přiřazením *b(u)* = *c*. Hodnotu proměnné, v níž máme číslo použité barvy, na začátku nastavíme na nulu (*B* = 0). Průběžně uplatňujeme tyto výběrové podmínky:

1. V grafu vyhledáme doposud neobarvené uzly. Je-li jich více, uplatníme následně další výběrové kritérium.
2. Mezi doposud neobarvenými uzly vyhledáme uzel, jehož již obarvení sousedé jsou obarveni největším počtem různých barev (nikoliv uzel, který má nejvíce již obarvených sousedů). Zřejmě na obarvení takového uzlu v současnosti zůstává nejmenší počet barev. Je-li takových uzlů více, uplatníme ještě další výběrové kritérium.

3. Mezi uzly vybranými v předchozím kritériu najdeme uzel, který má největší počet doposud neobarvených sousedů. Zřejmě pokud by byly následně obarveny různými barvami, bylo by obtížné najít volnou barvu pro obarvení tohoto uzlu. Proto tento uzel obarvíme přednostně.

Uzel, který byl předchozími kritérii vybrán jako následující uzel pro obarvení, označíme *u*. Najdeme nejnižší barvu *c* takovou, že jí není obarven žádný z již obarvených sousedů uzlu *u*. Uzel *u* touto barvou obarvíme $b(u) = c$. Je-li $c > B$ (*c* je nová, doposud nepoužitá barva), nastavíme proměnnou *B* na číslo této barvy $B = c$. Výběr uzlů opakujeme tak dlouho, dokud nejsou obarveny všechny uzly grafu. Na závěr je v proměnné *B* počet barev, jimiž je graf obarven.

Tento algoritmus je v programu pod položkou menu „Barvení (heuristicky). Protože nedostačuje k vyřešení všech Sudoku zadání, byl implementován další algoritmus.

4.1.2 Obecný algoritmus barvení grafu s backtrackingem

Jednotlivé barvy, které použijeme k barvení grafu, budeme značit přirozenými čísly (1, 2, …). Právě zkoumané částečné obarvení budeme uchovávat v hodnotách *B(x)*. Hodnoty *BARVA(x)* budou obsahovat nejlepší dosud nalezené obarvení. Proměnná *OMEZ* bude obsahovat číslo barvy, kterou již nechceme použít.

Při postupu vpřed dáme následujícímu vrcholu nejnižší možnou barvu, která není momentálně použita u vrcholů z množiny sousedů tohoto vrcholu. Nižší barvu není možno použít a vyšší barvy budou zkoušeny v dalším průběhu backtrackingu. Návraty v backtrackingu budeme provádět v situaci, kdy některý vrchol dostane barvu *OMEZ* nebo vyšší. Cílem návratu je snížení barvy ve vrcholu. Jelikož nižší barvy jsou již vyzkoušeny, je třeba zvýšit barvu v některém z předešlých vrcholů. Celý algoritmus lze zapsat následovně:

1. Inicializace: x = 1; $B(x) = 1$; $OMEZ = n + 1$.
2. Postup vpřed: $x = x + 1$.
   Jestliže $x > n$, pokračujeme krokem 3. V opačném případě položíme $B(x) = minimum$ z barev nevyskytujících se v množině sousedů.

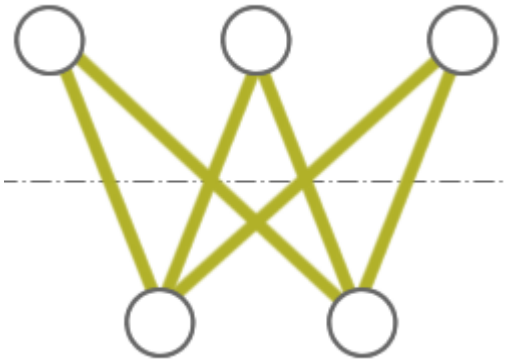
Jestliže $B(x) \geq OMEZ$, vracíme se na předchozí vrchol a pokračujeme krokem 4. Jinak opakujeme krok 2.

3. Pro všechny vrcholy provedeme $BARVA(x) = B(x)$ a $OMEZ$ = maximum z hodnot BARVA. Konec výpočtu.
4. Návrat: položíme b = minimum z barev větších než $B(x)$, které se nevyskytují v množině sousedů. Pokud je $b < OMEZ$, položíme $B(x) = b$ a pokračujeme krokem 2. Jinak jdeme na předchozí vrchol a opakujeme krok 4.

Při řešení hry Sudoku můžeme tento algoritmus ještě upravit, protože předem známe počet barev. Nemusíme tedy obarvovat graf více barvami a následně se pokoušet počet barev snížit, ale při kolizi barev provedeme návrat a změnu barvy v předchozích vrcholech. Tento algoritmus naleznete opět implementován ve výsledném programu pod položkou „Barvení (backtracking)“.

4.2 Tvorba nezávislých podmnožin grafu

Druhou možnou cestou řešení Sudoku za pomoci teorie grafů je rozdělení vrcholů na devět skupin, přičemž žádné dva vrcholy v rámci skupiny nejsou spojeny hranou. Takové grafy se v teorii grafů nazývají *k-partitní*. *Bipartitní* graf je možno definovat jako graf, jehož množinu vrcholů je možné rozdělit na dvě disjunktní množiny tak, že žádné dva vrcholy ze stejné množiny nejsou spojeny hranou. *K-partitní* graf je pak možno obdobně rozdělit do *k* skupin. Bipartitní graf je zobrazen na obrázku 18.



Obrázek 18 – Bipartitní graf (12)

Určení maximální nezávislé podmnožiny grafu je úloha patřící do třídy NP-úplných problémů. Není znám jiný postup než postupné procházení jednotlivých uzlů grafu a hledání největší nezávislé podmnožiny. Má-li graf *m* uzlů, pak je celkem 2^m podmnožin uzlů grafu. Tedy počet podmnožin roste exponenciálně s počtem uzlů grafu. Projít tyto podmnožiny v přijatelném čase je zvládnutelné jen pro grafy s nízkým počtem uzlů (řádově do 30 uzlů) (14). Při aplikaci této metody na řešení Sudoku úloh nehledáme maximální nezávislou podmnožinu, ale podmnožiny o devíti vrcholech. Tento problém je tedy zvládnutelný v reálném čase. Přesto se jedná o složitou úlohu, protože většina vrcholů může náležet současně do více podmnožin, přičemž existuje jediné správné rozdělení.

4.2.1 Obecný algoritmus tvorby nezávislých podmnožin grafu

Algoritmus je založen na myšlence, že větší nezávislá množina uzlů se sestaví snadněji z uzlů, které mají v grafu málo sousedů. Algoritmus vybere vždy uzel s nejnižším stupněm, přidá ho do podmnožiny a vyloučí tento uzel a všechny jeho sousedy z dalšího uvažování.

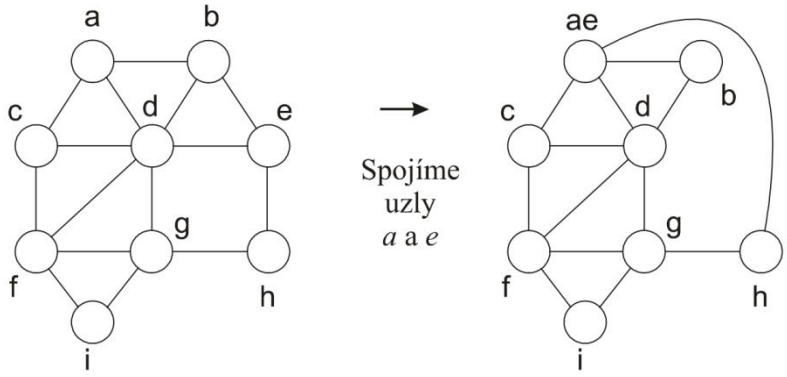
Při popisu algoritmu budeme výchozí graf značit *G*. Podgraf obsahující uzly, které lze ještě přidat k sestavované nezávislé podmnožině budeme značit *G'*. A *N* bude značena maximální nezávislá podmnožina uzlů. Průchod algoritmu bude vypadat následovně:

1. Počáteční podmínky: $N = 0$, sestavovaná nezávislá podmnožina je na začátku prázdná. $G' = G$, podgraf *G'* obsahující uzly, které lze ještě přidat k podmnožině, je na začátku celý výchozí graf.
2. Průběžný krok: v podgrafu *G'* najdeme uzel *u* s nejmenším stupněm. Je-li v něm více takových uzlů, vezmeme libovolný z nich. Uzel *u* přidáme k sestavované nezávislé podmnožině a z podgrafu *G'* odstraníme tento uzel a rovněž všechny uzly, které s ním sousedí.
3. Krok 2 provádíme tak dlouho, dokud není podgraf *G'* prázdný.

Jak je patrné, algoritmus obsahuje jistý stupeň náhodnosti zařazování vrcholů do podmnožin. Pro řešení úloh Sudoku je nezbytné takový algoritmus rozšířit o techniku backtrackingu. Tato metoda je ve vytvořeném programu pod názvem „Nezávislé množiny“.

4.3 Slučování vrcholů

Další technikou teorie grafů, kterou je možno použít k řešení Sudoku, je slučování vrcholů grafu. Slučují se vždy dva vrcholy, které spolu nesousedí. Výsledný vrchol je pak spojen hranami se všemi vrcholy, se kterými byly spojeny výchozí dva vrcholy. Postupným spojováním vznikne úplný graf. Úplný graf nelze obarvit jinak, než že se každému vrcholu přiřadí jiná barva. Stejnou barvou jsou pak obarveny i všechny vrcholy, ze kterých byl vrchol sloučen. Princip slučování je naznačen na obrázku 19.



Obrázek 19 – Slučování vrcholů (14)

4.3.1 Obecný algoritmus slučování vrcholů

Následující algoritmus aplikuje myšlenku slučování vrcholů a jejich následné obarvení:

1. Vytvoření úplného grafu: v grafu najdeme dva nesousední uzly *u* a *v*. Provedeme v grafu spojení těchto uzlů tak, že je nahradíme jedním uzlem, označme tento uzel *uv*. Nový uzel spojíme hranami se všemi původními sousedy uzlů *u* a *v*. Tento krok opakujeme tak dlouho, dokud lze spojovat nějaké dvojice uzlů. Po ukončení spojování dostaneme úplný graf.
2. Vlastní obarvení: vytvořený úplný graf obarvíme. Úplný graf nelze obarvit jinak, než že každému jeho uzlu přiřadíme samostatnou barvu. Následně zpětně spojené uzly začneme rozdělovat, přičemž vždy stejnou barvu jako má uzel *uv*, dáme rovněž uzlům *u* a *v*, jejichž spojením uzel *uv* vznikl. Proces rozpojování uzlů opakujeme tak dlouho, dokud se nedostaneme k původnímu grafu.

Tento algoritmus také obsahuje jistý stupeň náhodnosti spojování vrcholů s dalšími vrcholy. Pro řešení úloh Sudoku by bylo nezbytné takový algoritmus rozšířit opět o techniku backtrackingu. Jeho implementace by byla podobná jako u předchozích algoritmů s backtrackingem.

5 ANALÝZA

Pro vytvoření algoritmu, který řeší Sudoku za pomoci teorie grafů, je třeba kromě možnosti ručního vkládání zadání také generátor náhodných zadání. Ten usnadní práci, zvláště při testování. Je třeba ověřit správnost a jednoznačnost generovaného zadání, což umožní řešitel, založený na klasických algoritmech. Dále je nezbytné výsledky algoritmů vhodně prezentovat na obrazovce. Pro zobrazování je třeba znát datovou strukturu uchovávající Sudoku úlohu. Poslední částí je vhodná prezentace principu řešení.

5.1 Požadavky

Ze zadání diplomové práce vyplývají pouze požadavky na použití algoritmů známých z teorie grafů a na grafickou prezentaci řešení. V průběhu tvorby aplikace byly stanoveny další požadavky, které usnadňují práci s aplikací a zvyšují její použitelnost. Mezi tyto požadavky patří:

- generování náhodného zadání Sudoku s volbou počtu vodítek,
- vytváření vlastního zadání Sudoku,
- řešení Sudoku hádanky uživatelem,
- automatické řešení Sudoku hádanky různými algoritmy,
- ukládání a načítání hracího pole,
- kontrola řešení,
- jednoduché a příjemné uživatelské prostředí.

5.2 Programovací jazyk

Protože cílem práce není vytvoření co nejrychlejšího algoritmu, je vhodné použití objektově orientovaného jazyka. Volba OOP sice znamená vyšší režijní náklady a tedy delší čas potřebný pro výpočet, ale přináší celou řadu výhod. Mezi výhody patří zejména dědičnost, větší přehlednost zdrojového kódu a také možnost jeho opětovného použití. Vzhledem k povaze dané úlohy jsem zvolil programovací jazyk C#, který zaručuje poměrně rychlé

vykonávání příkazů a poskytuje také vhodné knihovny pro tvorbu uživatelského rozhraní a následnou grafickou prezentaci.

5.3 Datové struktury

Již z pohledu na pole Sudoku je patrné, že nejjednodušší způsob reprezentace hry v paměti počítače je použití dvourozměrného pole celých kladných čísel. Tato reprezentace je skutečně pro základní algoritmy řešící Sudoku dostatečná. Pro potřeby postupů z teorie grafů je vhodné navrhnout sofistikovanější datovou strukturu. Ta by měla uchovávat potřebná data a další informace nezbytné pro algoritmy pracující nad těmito datovými strukturami. Dále je vhodná správná volba použitých datových typů, což optimalizuje provádění operací s těmito typy (například vyhledávání).

5.4 Složitost řešení

Řešení obecné Sudoku matice patří mezi NP-úplné úlohy. NP úloha je množina problémů, které lze řešit v polynomiálně omezeném čase na nedeterministickém *turingově* stroji (na počítači, který umožňuje v každém kroku rozvětvit výpočet na *n* větví, v nichž se posléze hledá řešení současně). NP-úplné problémy jsou takové nedeterministicky polynomiální problémy, na které jsou polynomiálně redukovatelné všechny ostatní problémy z NP. To znamená, že třídu NP-úplných úloh tvoří ty nejtěžší úlohy z NP. Neexistuje deterministicky polynomiální algoritmus pro nějakou NP-úplnou úlohu, který by tuto úlohu řešil v polynomiálním čase. Proto je třeba řešit pouze Sudoku zadání s dostatečným počtem vodítek, která zaručují vyřešení Sudoku v polynomiálním čase. Přesto se může rychlost jednotlivých algoritmů značně lišit, a proto na jeho výběru může velmi záležet.

6 IMPLEMENTACE

V této kapitole se zaměřím na popis postupu při vytváření samotné aplikace, která je výsledkem této práce. Ze všeho nejdříve je třeba implementovat nějakého řešitele, který bude využit při generování zadání Sudoku. Při testování na generovaném zadání je teprve možné budovat řešitele Sudoku využívajícího techniky z teorie grafů. Další možností by bylo vložení pevného zadání nebo možnost zadávat hodnoty ručně, ovšem vedoucí k prodloužení další práce. Implementace více řešitelů může později vést k jejich porovnání a zhodnocení vhodnosti jednotlivých technik. Při dalším vysvětlování fungování aplikace budu používat některé pojmy z oblasti programování, které je nutné znát pro úplné pochopení všech postupů. Pro zjednodušení a zpřehlednění budu názvy metod uvádět bez vstupních parametrů a jiným typem písma, například `Metoda()`.

6.1 Třída Matice

Třída *Matice* poskytuje metody pro základní manipulaci s maticemi. Jedná se o statickou třídu, která neuchovává žádná data a pouze provádí dané operace. Tato třída je pouze pomocná a seskupuje takové obecné operace, které jsou využívány jinými třídami. Dále jsou stručně popsány konkrétní implementované operace:

- Metoda `Kopiruj()` pracuje se dvěma maticemi stejného typu a rozměrů. Cyklicky projde všechny políčka první matice a zkopíruje jejich hodnoty do odpovídajících políček matice druhé.
- Metoda `JsouStejne()` opět pracuje se dvěma maticemi stejného typu a rozměrů. Cyklicky prochází obě matice a porovnává jejich hodnoty. Pokud se některá neshoduje, vrací *false*, po porovnání všech hodnot vrací *true*.
- Metoda `Vymaz()` prochází zadanou matici a všechny její hodnoty nastaví na hodnotu *null*.
- Metoda `VseVyplneno()` prochází opět jednu zadanou matici. Pokud nalezne políčko, které nemá vyplněnou hodnotu, vrací *false*. Po projití všech políček vrací *true*.

- Metoda `Uloz()` slouží k uložení dané matice do souboru. Kromě dané matice je nutné zadat ještě cestu k souboru. Tento soubor je otevřen pro zápis nebo vytvořen, pokud neexistuje. Procházením matice je vytvořen textový řetězec, v němž jsou hodnoty ve sloupcích odděleny znakem středníku a hodnoty v řádcích jsou odděleny znakem pro nový řádek. Tento zápis odpovídá českému standardu formátu CSV, ve kterém je oddělovací čárka nahrazena středníkem. Celý řetězec je zapsán do otevřeného souboru.
- Metoda `Nacti()` opět potřebuje kromě matice, do které se hodnoty načtou, znát ještě cestu k souboru. Tento soubor je otevřen pro čtení. Při načítání probíhá opačný postup než při ukládání. Po přečtení znaku středníku se zapisuje hodnota do dalšího sloupce matice a po přečtení znaku nový řádek se zapisuje na další řádek matice. Počet hodnot v řádku není předem znám (prázdné hodnoty jsou *null*). Pro určení správné pozice je nutné ode všech znaků v řádku odečítat počet již načtených hodnot.

6.2 Třída Řešitel

Třída *Resitel* poskytuje metody pro řešení Sudoku zadání bez použití teorie grafů. Některé metody jsou využívány i následující třídou *Generator* k generování zadání Sudoku. Kromě dvou metod samotného řešení obsahuje třída ještě pomocné metody, které slouží k ověřování jednotlivých políček, případně k eliminaci kandidátů. Dále jsou popsány tyto pomocné operace:

- Metoda `MuzuVlozit()` zjistí pro konkrétní políčko hry Sudoku a navrhovanou hodnotu, zda je možné tuto hodnotu vložit na požadovanou pozici, aniž by byla porušena pravidla Sudoku. Pokud nejsou pravidla porušena (tzn., že není stejná hodnota již u některého ze sousedních vrcholů) vrací metoda *true*, při porušení pravidel vrací *false*.
- Metoda `SpravneVyplneno()` prochází celé pole Sudoku a s využitím předchozí metody zjistí, zda je možné do každého políčka vložit jeho hodnotu, pokud nejsou někde porušena pravidla Sudoku. Při nalezení porušení pravidel vrací *false*, při projití všech políček vrací *true*.

- Metoda `aktualizujKandidaty()` je volána při doplnění správné hodnoty do políčka. Účelem této operace je projití všech sousedních políček (políčka ve stejném řádku, sloupci a bloku) a odstranění právě vložené hodnoty ze seznamu jejich kandidátů. Je jasné, že tyto políčka již tuto hodnotu obsahovat nemohou, a proto ji můžeme vymazat ze seznamů kandidátů, což ulehčí další řešení.
- Metoda `muzuVlozitBlok()` zjistí, zda lze v bloku vložit hodnotu pouze na danou pozici. Přesněji řečeno, projde postupně všechna políčka ve stejném bloku, která ještě nejsou vyplněna. Pro každé políčko zjistí přítomnost dané hodnoty v seznamu kandidátů. Pokud některé políčko obsahuje v seznamu kandidátů danou hodnotu, vrací *false*. Jinak vrací *true* a je tedy možné hodnotu vložit.
- Metoda `muzuVlozitRadek()` zjistí, zda lze v řádku vložit hodnotu pouze na danou pozici. Přesněji řečeno, projde postupně všechna políčka ve stejném řádku, která ještě nejsou vyplněna. Pro každé políčko zjistí přítomnost dané hodnoty v seznamu kandidátů. Pokud některé políčko obsahuje v seznamu kandidátů danou hodnotu, vrací *false*. Jinak vrací *true* a je tedy možné hodnotu vložit.
- Metoda `muzuVlozitSloupec()` zjistí, zda lze ve sloupci vložit hodnotu pouze na danou pozici. Přesněji řečeno, projde postupně všechna políčka ve stejném sloupci, která ještě nejsou vyplněna. Pro každé políčko zjistí přítomnost dané hodnoty v seznamu kandidátů. Pokud některé políčko obsahuje v seznamu kandidátů danou hodnotu, vrací *false*. Jinak vrací *true* a je tedy možné hodnotu vložit.

6.2.1 Logický řešitel

Jako první možnost při implementaci jsem zvolil metodu doplňování jediného možného kandidáta. Dále implementoval další logická pravidla popsaná výše tak, aby byl algoritmus schopen řešit většinu Sudoku zadání. Výsledná Metoda `LogickyResitel()` nejprve projde celé pole Sudoku a naplní pro každé políčko seznam kandidátů. Poté prochází postupně všechna políčka celého pole Sudoku a vyhledává prázdná políčka. Pokud nalezne políčko s jediným kandidátem, je jeho hodnota okamžitě vložena. Při nalezení více možností se pokračuje dalším políčkem. Následuje pravidlo, podle kterého musí být v každé skupině každá hodnota právě jednou. Jinými slovy, pokud je hodnota kandidátem pouze jednoho políčka skupiny, musí do něho být vložena. Zjištění této informace je docíleno

využitím pomocných metod `muzuVlozitBlok()`, `muzuVlozitRadek()` a `muzuVlozitSloupec()`. Hodnota je vložena, pokud některá z těchto metod vrátí *true*. Takto se projde celé pole Sudoku. Pokud byla při tomto jednom průchodu vložena alespoň jedna hodnota, je důvod k opětovnému procházení celého pole Sudoku.

Nevýhodou algoritmu založeného na logickém řešení je zejména pracná implementace jednotlivých logických postupů. Při vyšším počtu těchto postupů se stává zdrojový kód také méně přehledným. Naopak výhodou je menší časová náročnost algoritmu. Dále fakt, že řešení je z principu jednoznačné. Implementací jednotlivých logických postupů lze řešit různé obtížnosti zadání, což lze později využít ke generování různých obtížností zadání Sudoku.

6.2.2 Brute force

Metoda hrubé síly (brute force) byla obecně popsána již v předchozí části textu. Konkrétní implementace je pozměněna, aby byl redukován počet procházených možností. V první části je ověřeno, zda již vyplněné hodnoty splňují pravidla Sudoku, jinak by tato metoda hledala řešení velmi dlouho a nikdy by ho nenalezla. Jedním z požadavků na tuto metodu je ověření, že neexistuje jiné řešení daného zadání Sudoku, což dobu vykonávání algoritmu také prodlužuje.

Samotný algoritmus prochází postupně celé pole Sudoku a vyhledává nevyplněná políčka. Do nalezeného políčka se pokouší vložit postupně hodnoty 1 až 9 s využitím metody `MuzuVlozit()`. Je vložena první hodnota, kterou vložit lze. Ta je uložena do seznamu již zkoušených hodnot, který se vede zvlášť pro každé políčko. Postupuje se na další políčko. Pokud nebyla vložena žádná hodnota, postupuje se na předchozí políčko. Zde se pokouší vložit jinou hodnotu, která není v seznamu zkoušených hodnot. Políčka, jejichž hodnoty byly určeny již v zadání, se samozřejmě neupravují. Když algoritmus dojde na poslední políčko, je nalezeno jedno řešení. Algoritmus však nekončí a zvyšováním předešlých hodnot se snaží nalézt další řešení. Algoritmus končí po nalezení druhého řešení (Při nalezení druhého řešení je jasné, že zadání není jednoznačné a není důvod pokračovat.), nebo po projití všech možností. Poté je vrácen počet nalezených řešení.

Výhodou metody hrubé síly je obecnost algoritmu (možné použití i pro jiná *n*) a jeho relativní jednoduchost. Další výhodou je fakt, že algoritmus je vždy schopen nalézt řešení (pokud existuje). Nevýhodou je nutnost dalšího výpočtu k ověření jednoznačnosti zadání. To ještě zvyšuje již tak vysoké časové nároky algoritmu. Doba výpočtu se může pro různá zadání značně lišit.

6.3 Třída Generátor

Ke generování Sudoku zadání lze použít dva odlišné postupy, které lze odlišit podle způsobu vytváření zadání:

1. Generátor založený na přidávání prvků vychází z prázdného pole Sudoku a jsou náhodně generovány pozice a hodnoty prvků, které se mají vložit. Následně je ověřeno, zda po vložení tohoto nového prvku bude zadání nadále splňovat pravidla Sudoku. Je určeno, kolik takových prvků se má vygenerovat (minimálně sedmnáct). Poté je využit nějaký algoritmus řešení pro ověření, že lze zadání vyřešit a že má právě jedno řešení.
2. Generátor založený na odebírání prvků z kompletního pole Sudoku naopak nejdříve vygeneruje úplné pole, které splňuje pravidla hry Sudoku. Poté jsou náhodně vybírány pozice, ze kterých se pokusí hodnotu odebrat. S využitím některého algoritmu pro řešení se ověří řešitelnost a jednoznačnost nového pole. Pokud je řešitelné, pokračuje se dalším odebíráním, jinak se hodnota vrátí zpět a vybere se náhodně jiný prvek. Vše se opakuje, dokud není odebráno požadované množství hodnot. Původní kompletní pole se u tohoto typu generátoru stává zároveň řešením pro danou Sudoku úlohu.

Oba typy generátorů dávají srovnatelné výsledky, a proto nebyla při výběru preferována konkrétní metoda. Ve své práci jsem implementoval druhý typ generátoru (založený na odebírání prvků). Pro tento typ je nutné nejprve generovat úplné pole Sudoku a poté z něj postupně odebírat prvky. To vše zajišťuje třída *Generator*, která navíc pro ověření řešitelnosti využívá třídu *Resitel*. Kromě toho ještě třída uchovává informaci o počtu odebíraných prvků a parametr nastavující symetričnost zadání, které se inicializují při konstrukci generátoru.

6.3.1 Generátor úplného pole Sudoku

Metoda `GenerujReseni()` nejprve vymaže předchozí řešení (s využitím metody `Vymaz()` třídy *Matice*). Poté vytvoří dva seznamy typu *Blok*. Jeden seznam pro bloky, které je nutné vyplnit. Druhý pro bloky, které jsou již vyplněny. *Blok* obsahuje pouze jednoznačné souřadnice identifikující ho v poli Sudoku. Z nevyplněných bloků se náhodně vybere jeden. Do tohoto bloku se generují jednotlivé hodnoty políček tak, aby splňovaly pravidla Sudoku. To je ověřováno pomocí metody `MuzuVlozit()` třídy *Resitel*. Pokud nelze doplnit všechny hodnoty bloku, začne se vyplňovat celý blok znovu. Při vyplnění všech políček je blok vložen do vytvářeného řešení a pokračuje se dalším blokem. Pokud nelze vložit žádný další blok, je vybrán některý z již vyplněných a je vymazán. Algoritmus pokračuje, dokud není vloženo všech devět bloků.

6.3.2 Generátor zadání

Metoda `GenerujZadani()` může pracovat hned v několika režimech. Je jí předán parametr určující, který typ řešení se má použít k ověřování správnosti. Dále pracuje s parametrem, který určuje, zda bude výsledné zadání symetrické. Posledním parametrem je počet prázdných políček, který má generované zadání obsahovat.

Metodě je parametrem předáno úplné pole Sudoku vygenerované předchozí metodou. To je zkopírováno (metodou `Kopiruj()` třídy *Matice*) pro práci a ověřování. Náhodně je vybráno políčko, které ještě není prázdné. Hodnota tohoto políčka je vymazána. V případě volby symetrického zadání je vymazána hodnota i na políčku se symetrickými souřadnicemi ($9 - x$ a $9 - y$). Následně se pokusí algoritmus toto nově vzniklé pole vyřešit jednou z metod řešení, která je určena v parametru metody. Při úspěšném vyřešení pokračuje odebráním dalšího prvku. Pokud se řešení nezdaří, je hodnota prvku vrácena. Případně se vrátí hodnota některého dalšího políčka. V případě generování symetrického zadání se samozřejmě vracejí i prvky symetrické. Algoritmus je ukončen ve chvíli, kdy obsahuje požadovaný nebo vyšší počet prázdných políček.

6.4 Třída Vrchol

Políčko hry Sudoku bude reprezentováno samostatnou třídou nazvanou *Vrchol*, která bude uchovávat:

- hodnotu buňky,
- číslo vrcholu,
- seznam sousedních vrcholů,
- seznam barev použitých u sousedních vrcholů,
- seznam již zkoušených barev na této buňce,
- informaci, zda byla tato buňka vyplněna již v zadání.

Je vhodné, aby hodnota buňky byla reprezentována jako nulovatelný typ (umožňující nastavit hodnotu na *null*), což umožní jednoznačně říci, zda je daná buňka vyplněna či nikoliv. Číslo vrcholu nabývá hodnot z intervalu 0 až 80. V číslování se postupuje zleva doprava, shora dolů. Seznam sousedních vrcholů obsahuje čísla vrcholů, se kterými daný vrchol sousedí. Těchto sousedů je právě dvacet. Tuto hodnotu dostaneme sečtením devíti políček v řádku, devíti políček ve sloupci, devíti políček v bloku a následným odstraněním duplicit. Seznam barev sousedních vrcholů uchovává informaci o barvách, kterými jsou již obarveny některé sousední vrcholy. Jinými slovy barvy, které již nemohou být pro daný vrchol použity. Do seznamu již zkoušených hodnot jsou ukládány hodnoty barev, které nevedly k řešení a je tedy zbytečné je zkoušet znovu. Poslední atribut udává, že je daná buňka zadáním a není ji možno měnit.

Kromě vlastností (properties) umožňující čtení a změnu jednotlivých atributů obsahuje třída ještě metodu `ZjistiPocetBarevSousedu()`. Tato metoda projde všechny sousední vrcholy daného vrcholu. Barva je přidána, pokud je sousední vrchol obarven a zároveň není jeho barva obsažena v seznamu barev použitých u sousedních vrcholů zkoumaného vrcholu. Seznam barev tedy obsahuje každou barvu použitou u sousedních vrcholů právě jednou.

6.5 Třída Graf

Graf je pak reprezentován jako dvourozměrné pole tříd *Vrchol*. Tato reprezentace je velmi přehledná zvláště vzhledem k podobnosti se samotným polem Sudoku. Převod čísla vrcholu na dvourozměrnou souřadnici je triviální. Pozici v řádku získáme jako zbytek po celočíselném dělení čísla vrcholu devíti. Pozici ve sloupci pak získáme jako hodnotu celočíselného dělení čísla vrcholu devíti.

Třída Graf obsahuje kromě samotných algoritmů řešících Sudoku ještě tyto pomocné metody:

- Metoda `dejHodnoty()` vrací z grafové struktury pouze dvourozměrné pole hodnot.
- Metoda `pridejSousedniVrcholy()` projde políčka ve stejném řádku, sloupci, bloku a přidá pozici políčka do seznamu sousedních vrcholů. Procházení je prováděno, aby byl každý sousední vrchol vložen do seznamu právě jednou. Výsledný seznam má tedy vždy dvacet prvků.
- Metoda `vymazSousedniVrcholy()` odebere políčka ve stejném řádku, sloupci a bloku z grafu. Ve skutečnosti nastaví jejich hodnotu pouze na nulu a tím je vyřadí z dalšího uvažování.
- Metoda `aktualizujSousedniVrcholy()` nastaví hodnoty obsahující nulu zpět na *null*, aby je bylo možné opět uvažovat při dalším výběru.
- Metoda `vytvorGraf()` prochází pole reprezentující Sudoku a do každého políčka vloží nový vrchol, který obsahuje číslo vrcholu a seznam sousedních vrcholů. Tato metoda je volána při inicializaci grafu.
- Metoda `VyplnBarvyGrafu()` prochází pole obsahující vrcholy a nastaví barvu každého z nich na barvu převzatou z pole Sudoku.
- Metoda `GenerujSkript()` prochází všechny vrcholy grafu a do textového řetězce vypisuje spojení s tímto vrcholem a jeho sousedními vrcholy tak, aby zde nebyly duplicitní záznamy. Tento textový řetězec je vhodný jako vstup pro program *GraphViz*, který daný graf vykreslí.

6.6 Algoritmy teorie grafů řešící Sudoku

V této části jsou popsány upravené algoritmy teorie grafů pro řešení Sudoku úloh tak jak jsou implementovány ve vytvořené aplikaci.

6.6.1 Heuristické barvení

Algoritmus nejprve projde všechny vrcholy v grafu. Pro neobarvené vrcholy zjistí počet barev sousedů (bez duplicit). Tyto vrcholy jsou vkládány do seznamu, který je následně seřazen sestupně podle počtu barev sousedů. Vrcholu s nejvyšším počtem barev sousedů je přiřazena barva, která ještě není u sousedních vrcholů použita. Pokud existuje více takových barev, algoritmus není schopen rozhodnout a končí neúspěšným řešením. Po vložení alespoň jednoho prvku se celý proces opakuje.

6.6.2 Backtracking barvení

Pro vyřešení všech Sudoku zadání je nutné algoritmus barvení rozšířit o zpětné procházení (backtracking). Pro zjednodušení zpětného procházení je vhodné procházet graf postupně a ne od vrcholů s nejvyšším počtem barev sousedů.

Algoritmus prochází postupně vrcholy, tak jak jdou za sebou i políčka hracího pole Sudoku. Vrcholy s hodnotou vyplněnou v zadání se přeskakují. Pro aktuální vrchol se zjistí počet barev, kterými jsou obarveny sousední vrcholy. Vrchol je obarven první barvou, která není v seznamu barev, kterými jsou obarveny sousední vrcholy, a která není v seznamu již zkoušených hodnot pro tento vrchol. Pokud není nalezena vhodná barva, provádí se backtracking. Zpětné procházení spočívá ve vymazání hodnoty a vymazání seznamu již zkoušených hodnot pro aktuální vrchol. Dále se postupuje zpět na první vrchol, jehož hodnota nebyla určena v zadání. Jeho hodnota je vymazána a algoritmus se snaží najít pro tento vrchol další vhodnou barvu. Po obarvení všech vrcholů je nalezeno řešení.

6.6.3 Nezávislé množiny

Jedná se o další metodu, které je schopná řešit všechna Sudoku zadání jen po rozšíření o backtracking. Dělení grafu na nezávislé množiny a následný backtracking je možné realizovat velice podobně jako barvení grafu. Pro budoucí porovnání algoritmů je vhodné použít odlišný způsob, který se více blíží obecnému postupu dělení grafu.

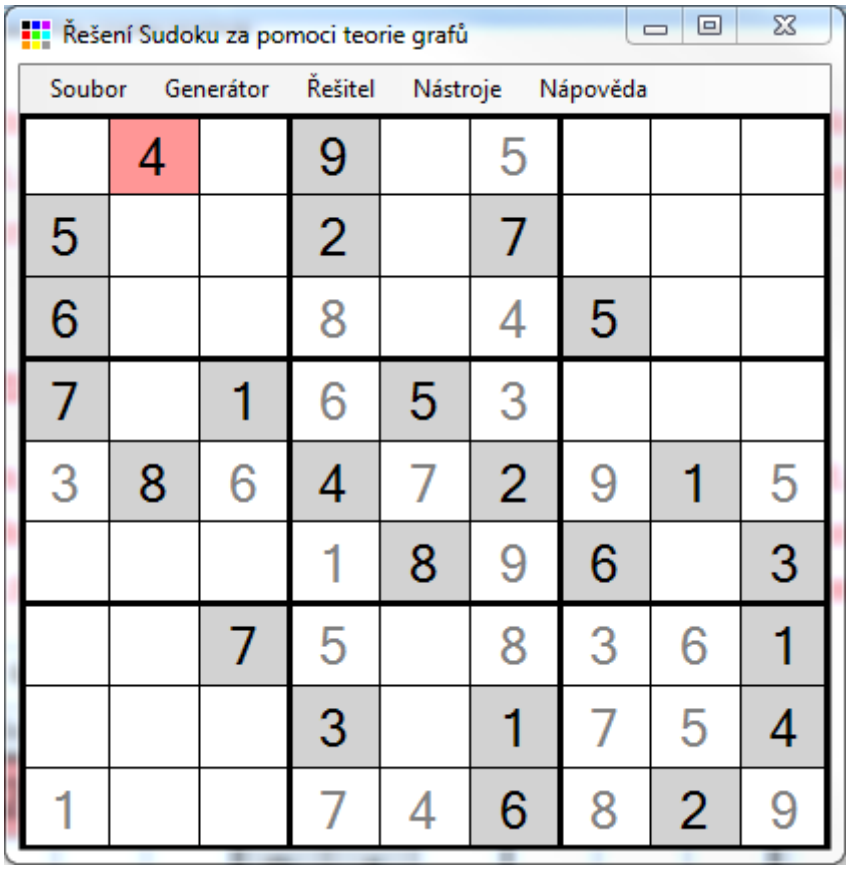
Algoritmus rozděluje vrcholy postupně do podmnožin 1 až 9. Nejprve tedy najde vrcholy s hodnotou jedna a vymaže všechny sousední vrcholy tohoto vrcholu (tyto vrcholy již nemohou patřit do stejné množiny). Poté prochází jednotlivé sloupce a snaží se nalézt vrchol, pro který ještě nebyla hodnota zkoušena a který má hodnotu *null*. Tento vrchol je možné zařadit do množiny. Po přiřazení jednoho vrcholu z každého sloupce se hledají vrcholy do další množiny. Pokud není po projití sloupce přiřazen vrchol do množiny, provádí se backtracking. Z množiny se vybere poslední vrchol, který je odebrán. Následuje snaha o zařazení dalšího prvku ze stejného sloupce. Pokud jsou z množiny odebrány všechny vrcholy, přejde se na předchozí množinu. Z této množiny jsou opět odebírány prvky a nahrazovány jinými ze stejného sloupce. Algoritmus končí po vytvoření devíti množin o devíti vrcholech.

7 UŽIVATELSKÉ ROZHRANÍ

V této části textu bude krátce popsáno vytvořené uživatelské rozhraní a práce s ním. V dnešní době je samozřejmostí vytvoření grafického uživatelského rozhraní, které umožňuje interaktivní ovládání programu (například pomocí myši).

7.1 Okno programu

Hlavní částí okna programu, které můžete vidět na obrázku 20, je samozřejmě samotná mřížka hry Sudoku. Mřížka je rozdělena dle zvyklostí a dále graficky rozlišuje hodnoty ze zadání a hodnoty řešené. Zbytek okna tvoří pouze menu sloužící k ovládání programu. Mřížka je zobrazována jako obrázek, který se překresluje stisknutím klávesy, kliknutím tlačítka myši, změnou velikosti okna programu, nebo spuštěním některé položky z menu.



Obrázek 20 – Okno programu

7.2 Vykreslování

K vykreslování herní mřížky slouží samostatná třída *Grafika* a její metoda `Kresli()`. V této metodě se nejprve projde celé pole Sudoku a hodnoty ze zadání se vykreslí s šedým pozadím. Vykreslují se jednotlivé obdélníčky, jejichž velikost je dána šířkou a výškou celé mřížky dělenou devíti. Dále se vykresluje barva pozadí aktivního políčka. Barva pozadí tohoto políčka je červená, pokud je určeno v zadání, jinak zelená. Následuje vykreslení hlavní mřížky silnějším tahem, které je realizováno jako vykreslení obdélníku přes celou mřížku, dvou vertikálních a dvou horizontálních linek v třetinových šířkách a výškách. Vedlejší mřížka je realizována vykreslením vertikálních a horizontálních linek s odstupem výšky a šířky mřížky dělené devíti. Nakonec je znovu projito pole Sudoku a jsou zapisovány hodnoty. Hodnoty zadání jsou vykreslovány vždy černou barvou. Řešené hodnoty jsou vykreslovány šedou, zelenou nebo červenou barvou. Barva závisí na volbě zobrazování správnosti vyplnění. Velikost písma, stejně jako velikost jednotlivých políček, je počítána z celkové velikosti mřížky a je tedy možné měnit velikost okna aplikace a zobrazení se přizpůsobí.

7.3 Ovládání

V mřížce se lze pohybovat kliknutím tlačítka myši na požadované políčko nebo s využitím kurzorových kláves na klávesnici. Samotné hodnoty jednotlivých políček se vkládají pomocí numerických kláves (alfanumerických také). Pro mazání zapsané hodnoty lze použít klávesy „nula“, „delete“, „backspace“ a „mezerník“. Tímto způsobem je možné vkládat do programu vlastní zadání, nebo řešit zadání generované programem. Ostatní operace se ovládají přes menu programu, nebo klávesovými zkratkami.

7.4 Menu

Všechny operace implementované v programu jsou pro zpřehlednění rozděleny do několika skupin, které dohromady tvoří ovládací menu programu. Menu je tedy rozděleno na podmenu *Soubor*, *Generátor*, *Řešitel*, *Nástroje* a *Nápověda*.

V podmenu *Soubor* je:

- Položka *Nový* slouží k vymazání stávajícího hracího pole i zadání.
- Položka *Otevřít* vyvolá standardní dialogové okno pro otevření souboru s přednastaveným typem souboru na CSV.
- Položka *Uložit* vyvolá standardní dialogové okno pro uložení souboru s přednastaveným typem souboru na CSV.
- Položka *Konec* ukončí program.

Z podmenu *Generátor* lze zavolat dva implementované generátory zadání, a to:

- Položkou *Generátor (brute force)* je zavolán algoritmus generování na základě řešení metodou hrubé síly.
- Položkou *Generátor (logicky)* je zavolán algoritmus generování na základě logického řešení.

Z podmenu *Řešitel* lze zavolat všechny implementované algoritmy řešení, a to:

- Položkou *Logický řešitel* je zavolán algoritmus logického řešení.
- Položkou *Brute force* je zavolán algoritmus řešení pomocí hrubé síly.
- Položkou *Barvení (heuristicky)* je zavolán heuristický algoritmus barvení grafu.
- Položkou *Barvení (backtracking)* je zavolán algoritmus barvení grafu využívající backtracking.
- Položkou *Nezávislé množiny* je zavolán algoritmus dělící vrcholy grafu do nezávislých množin.

V podmenu *Nástroje* je:

- Položka *Vymaž řešení* zachová zadání a vymaže pouze hrací pole.
- Položka *Kontrola řešení* zavolá metodu kontrolující pravidla Sudoku a zobrazuje informaci o správném vyplnění, špatném vyplnění nebo nevyplnění hodnot.
- Položka *Skript grafu* zavolá metodu generující skript pro program *GraphViz* a tento skript zobrazí v dialogovém okně.

- Položka *Nastavení* vyvolá nové dialogové okno, ve kterém je možné nastavit počet generovaných hodnot, symetričnost generovaného zadání a zobrazování správnosti vyplnění hodnot v poli. Všechna tato nastavení jsou ukládána do konfiguračního souboru, ze kterého jsou aplikací opětovně načítána.

Podmenu *Nápověda* obsahuje pouze položku *O aplikaci*, která zobrazí informace o programu (verzi programu, jméno autora a popis programu).

7.5 Použití programu

Po spuštění aplikace musí uživatel nejprve určit zadání, které bude řešeno. Aplikace umožňuje uživateli:

- Vytvořit zadání vepsáním hodnot do mřížky. V mřížce se pohybuje pomocí myši nebo kurzorových kláves. Hodnotu pak zapisuje do každého políčka pomocí klávesnice.
- Generovat zadání samotným programem. Vyvoláním položky menu *Generátor –> Generovat (brute force),* nebo *Generátor –> Generovat (logicky).* Generované zadání je možné ovlivnit v dialogovém okně, které lze vyvolat kliknutím na položku menu *Nástroje –> Nastavení.* Zde může uživatel nastavit počet prázdných polí v zadání a to, zda bude zadání symetrické.
- Otevřít zadání ze souboru. Zavoláním položky menu *Soubor –> Otevřít* a vybráním souboru se zadáním ve formátu CSV.

Vytvořené zadání může uživatel uložit do souboru CSV pro další použití zavoláním položky menu *Soubor –> Uložit*.

Druhým krokem při práci s aplikací je samotné řešení. Aplikace nabízí uživateli několik možností řešení Sudoku úloh:

- Uživatel může řešit úlohu vpisováním hodnot do herní mřížky pomocí klávesnice.
- Uživatel může využít algoritmů aplikace vyvoláním některé položky z podmenu *Řešitel*.

Následně může uživatel zavoláním položky menu *Nástroje –> Vymaž řešení* vymazat pouze řešení úlohy a řešit stejnou úlohu jinou metodou.

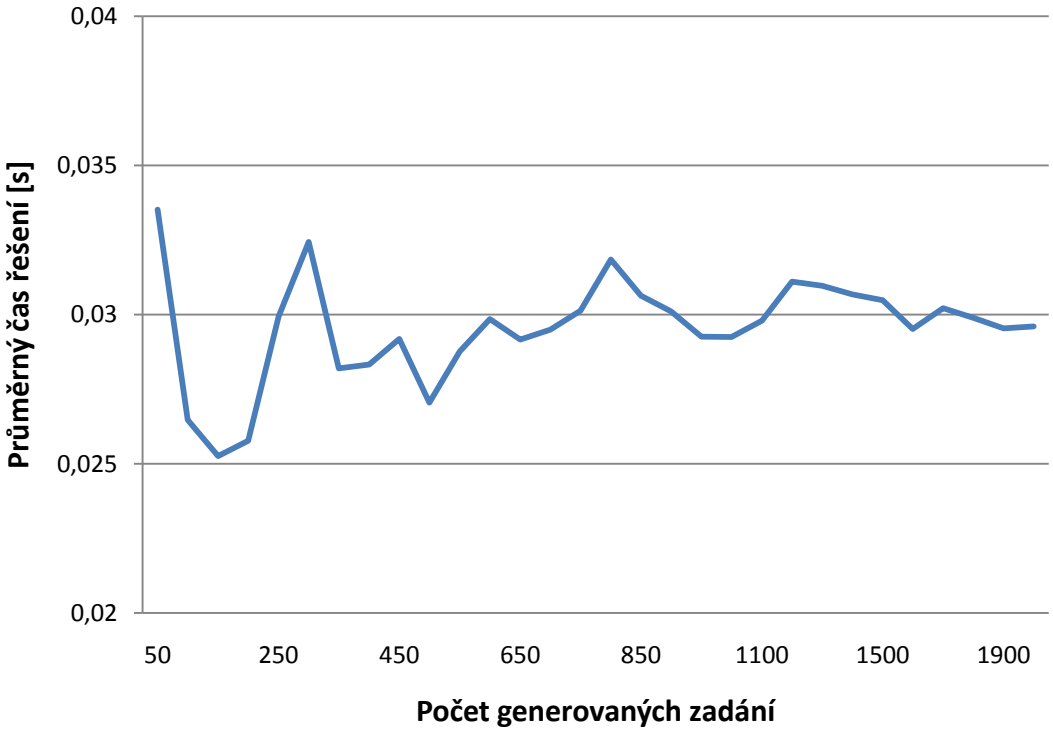
Dalším volitelným krokem je kontrola řešení. Pro generovaná zadání je možné v dialogovém okně *Nastavení* (vyvolá se kliknutím na položku menu *Nástroje –> Nastavení*) zatrhnout volbu *Zobrazovat správnost vyplnění*, která zajistí zobrazení správných hodnot zelenou barvou a špatných červenou. Ověření správnosti řešení všech zadání je možné zavoláním položky menu *Nástroje –> Kontrola řešení*. Kromě informace o správnosti řešení je zobrazena i doba řešení.

Pro vizualizaci aktuálního Sudoku pole formou grafu klikne uživatel na položku menu *Nástroje –> Skript grafu*, která vyvolá dialogové okno se skriptem. Tento skript je pak nutné použít jako vstup programu *GraphViz*.

8 POROVNÁNÍ ALGORITMŮ

Porovnání implementovaných algoritmů bude demonstrováno na několika převzatých zadáních a dále na zadáních generovaných vestavěnými generátory. Algoritmus založený na heuristickém barvení je možné z porovnání vyřadit, protože většinu zadání není schopen vyřešit. První sledovanou veličinou bude čas běhu řešícího algoritmu. Nebude se tedy započítávat načtení nebo generování zadání. Druhou sledovanou veličinou bude počet průchodů algoritmu vedoucí k úspěšnému vyřešení.

Testování bude prováděno na osobním počítači s dvoujádrovým procesorem pracujícím na frekvenci 2,8 GHz. Experimentálně bylo vyvráceno ovlivňování doby výpočtu jinými běžícími nenáročnými procesy, proto je dostatečné převzatá zadání řešit pouze jednou. Jak vyplývá z grafu na obrázku 21, průměrný čas řešení jednoho zadání se ustaluje s rostoucím počtem generovaných zadání. Při generování 2 000 zadání se průměrná doba řešení mění méně než v řádu milisekund. Taková přesnost je dostatečná, a proto bude testování generovaných zadání opakováno právě 2 000 krát.



Obrázek 21 – Graf závislosti průměrné doby řešení na počtu generovaných zadání

První testované zadání je zobrazeno na obrázku 22. Jedná se o těžké zadání pro metody systematického procházení. Pro lidského řešitele je toto zadání lehké. Proto algoritmus založený na doplňování hodnot na základě metod odvozených od lidského řešitele bude toto zadání řešit rychle.

							1	
4								
	2							
				5		4		7
		8				3		
		1		9				
3			4			2		
	5		1					
			8		6			

Obrázek 22 – První zadání pro testování (4)

Výsledky prvního testování jsou uvedeny v tabulce 1. Z výsledků je patrné, že systematické metody řešení musejí projít velké množství možností doplnění hodnot do mřížky, než naleznou řešení. Nejlepšího výsledku dosáhl algoritmus založený na logickém řešení, naopak nejhoršího výsledku dosáhl algoritmus barvení grafu s backtrackingem.

Tabulka 1 – Výsledky testování prvního zadání

	Čas vykonávání [s]	Počet průchodů
Logický řešitel	0,004	9
Brute force	168,912	192 782 279
Barvení (backtracking)	273,747	60 621 718
Nezávislé množiny	131,006	723 862

Druhé testované zadání je zobrazeno na obrázku 23. Toto zadání není tolik obtížné pro metody systematického procházení, ale je obtížné pro lidského řešitele. Jelikož nejsou implementovány všechny postupy lidského řešitele, nepodařilo se algoritmu na nich založených toto zadání vyřešit.

8								
				6			3	
1		2	4					
5					7			
	3						6	
			1					2
					5	2		1
	8			7				
								4

Obrázek 23 – Druhé zadání pro testování (3)

Výsledky testování jsou shrnuty v tabulce 2. Logický řešitel je z důvodu neúspěšného řešení zcela vynechán. Nejlepšího výsledku dosáhl algoritmus dělící vrcholy na nezávislé množiny. Nejpomalejšího času pak dosáhl opět algoritmus barvení grafu.

Tabulka 2 – Výsledky testování druhého zadání

	Čas vykonávání [s]	Počet průchodů
Brute force	4,402 344	5 054 778
Barvení (backtracking)	10,785 156	2 090 902
Nezávislé množiny	0,062 500	139

Následuje testování na generovaných zadáních. Výsledky algoritmů řešících zadání generované na základě logického řešitele jsou uvedeny v tabulce 3. Zadání byla generována na základě logického řešitele, a tudíž dosahuje logický řešitel i nejlepších výsledků. Nejdelší průměrné doby řešení dosáhl tentokrát algoritmus dělící vrcholy na nezávislé množiny.

Tabulka 3 – Výsledky testování zadání generovaných na základě logického řešitele

	Průměrný čas vykonávání [s]	Průměrný počet průchodů
Logický řešitel	0,000 315	5,851
Brute force	0,029 435	36 136,250
Barvení (backtracking)	0,084 550	18 291,643
Nezávislé množiny	0,437 522	2 996,796

Poslední testování bylo uskutečněno na zadáních generovaných generátorem založeném na řešiteli Brute force. Výsledky testování jsou uvedeny v tabulce 4. Logický řešitel nebyl schopen vyřešit všechna vygenerovaná zadání, a proto není uveden. Ostatní výsledky kopírují ty předešlé, jen jsou touto metodou generovány složitější zadání, a proto je potřebný čas i počet průchodů algoritmem o něco vyšší.

Tabulka 4 – Výsledky testování zadání generovaných na základě řešitele Brute force

	Průměrný čas vykonávání [s]	Průměrný počet průchodů
Brute force	0,032 830	40 612,528
Barvení (backtracking)	0,093 526	19 859,539
Nezávislé množiny	0,489 013	3 177,477

V průběhu testování bylo zjištěno, že rychlost řešení jednotlivými algoritmy je především závislá na konkrétním zadání. Zadání, které je vyřešeno jednou metodou rychle, je řešeno jinou metodou daleko delší dobu a naopak. Z průměrných dob řešení získaných při řešení generovaných zadání lze snadno odvodit rychlost jednotlivých algoritmů. Nejrychlejším algoritmem je, dle očekávání, algoritmus založený na lidském řešiteli. Aby byl tento algoritmus schopen řešit všechna zadání, bylo by nutné ho rozšířit o další metody řešení. Nejrychlejším algoritmem, který je schopen řešit všechna zadání, je algoritmus Brute force. Ten je následován algoritmem barvení grafu. Algoritmus založený na dělení vrcholů grafu na nezávislé množiny je o celý řád pomalejší, což lze vysvětlit zejména režijními operacemi, které jsou dány zcela odlišným přístupem. Celkově lze naměřené časy, které v průměru nepřesahují půl sekundy, označit za uspokojivé.

Závěr

V průběhu práce jsem blíže popsal hru Sudoku a postupy řešení používané člověkem. Tyto znalosti mohou být využity k běžnému řešení Sudoku. Z programátorského hlediska jsou zajímavější techniky systematického procházení. Jedná se o nenáročné postupy, které vždy vedou ke správnému vyřešení zadání. Do této kategorie lze zařadit i upravené techniky teorie grafů.

V programu jsem implementoval dva druhy generátorů generující jednoznačná zadání. Dále jsem implementoval pět různých algoritmů řešících Sudoku. Po jejich porovnání na vzorku náhodných zadání lze konstatovat vhodnost použití technik teorie grafů pouze z hlediska náročnosti jejich implementace. Z hlediska doby výpočtu dosahují v průměru horších výsledků než ostatní zkoumané algoritmy. Nejrychlejšího zpracování lze, dle očekávání, dosáhnout algoritmy založenými na lidském postupu řešení. Ty je ovšem složité implementovat, aby byly schopny řešit všechna Sudoku zadání. Práce ale ukazuje, že metody teorie grafů lze prakticky k řešení Sudoku využít.

Grafická reprezentace je v programu řešena zobrazením mřížky Sudoku. Graf reprezentující Sudoku je možné zobrazit programem *GraphViz* generovaným skriptem.

Použitá literatura

1. Sudoku - Historia. *Sudoku.estranky.sk.* [Online] 2009. [Citace: 24. dubna 2010.] http://www.sudoku.estranky.sk/stranka/historia.
2. Sudoku. *Wikipedia.* [Online] 22. dubna 2010. [Citace: 24. dubna 2010.] http://en.wikipedia.org/wiki/Sudoku.
3. *Taking Sudoku Seriously.* **Taalman, Laura.** 2007, Math Horizons, stránky 5-9.
4. **Herzberg, Agnes M. a Murty, M. Ram.** Sudoku Squares and Chromatic Polynomials. *Notices of the AMS.* 2007.
5. Kakuro. *Wikipedia.* [Online] 25. listopadu 2009. [Citace: 24. dubna 2010.] http://en.wikipedia.org/wiki/Kakuro.
6. Killer sudoku. *Wikipedia.* [Online] 25. února 2010. [Citace: 24. dubna 2010.] http://en.wikipedia.org/wiki/Killer_sudoku.
7. Sudoku Cube. *Wikipedia.* [Online] 19. března 2010. [Citace: 24. dubna 2010.] http://en.wikipedia.org/wiki/Sudoku_Cube.
8. Metody řešení sudoku. *Jiven.* [Online] 20. prosince 2008. [Citace: 24. dubna 2010.] http://www.volny.cz/jiven/.
9. Řešení hrubou silou. *Wikipedia.* [Online] 21. května 2009. [Citace: 24. dubna 2010.] http://cs.wikipedia.org/wiki/Řešení_hrubou_silou.
10. Algorithmics of sudoku. *Wikipedia.* [Online] 19. dubna 2010. [Citace: 24. dubna 2010.] http://en.wikipedia.org/wiki/Algorithmics_of_sudoku.
11. **Sajdl, Václav.** *Software pro řešení hlavolamu Eternity 2 s použitím algoritmů prohledávání stavového prostoru.* Praha : České vysoké učení technické, 2009.
12. **Jirovský, Lukáš.** *Vybrané problémy z teorie grafů.* Praha : Univerzita Karlova, 2008. str. 84.
13. Teorie grafů. *Wikipedie.* [Online] 13. března 2010. [Citace: 24. dubna 2010.] http://cs.wikipedia.org/wiki/Teorie_grafů.
14. **Večerka, Arnošt.** *Grafy a grafové algoritmy.* Olomouc : Univerzita Palackého, 2007.
15. **Demel, Jiří.** Grafy a jejich aplikace. Praha : Academia, 2002. 1, str. 257. 80-200-0990-6.