ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

KATEDRA TECHNOLOGIÍ A MĚŘENÍ

DIPLOMOVÁ PRÁCE

Výukový program pro řešení přiřazovacího problému tzv. Maďarskou metodou

Miroslav Turek **2015**

Kopie zadání diplomové práce

ABSTRAKT

Předkládaná diplomová práce se zabývá problematikou distribučních úloh, přesněji přiřazovacím problémem řešeným Maďarskou metodou a okružním dopravním problémem („problém obchodního cestujícího“). Cílem této práce je praktické zpracování daných úloh v jazyce C++. Přiřazovací problém formou výukového programu, pomocí kterého by měl být uživatel schopen pochopit a naučit se dané problematice. Druhá část programu se zaměřuje na eliminaci parciálních smyček u okružního dopravního problému vznikajících při řešení obecnou Maďarskou metodou.

KLÍČOVÁ SLOVA

Maďarská metoda, okružní dopravní problém, přiřazovací problém, NP-úplný, minimalizace, maximalizace, účelová funkce, spotřebitel, dodavatel, penalizace, permutace, heuristika, metaheuristika, exaktní algoritmy, algoritmus, polynom, výukový program

ABSTRACT

The present thesis deals with the distribution problems, more specificly assignment problem to be solved by Hungarian method and traveling salesman problem. The aim of this work is the practical solution of these tasks in C ++. Assignment problem through the educational program thanks to which the user should be able to understand and learn the issue. The second part of the program focuses on the elimination of partial loops in traveling salesman problem arising in dealing with the general Hungarian method.

KEY WORDS

The Hungarian method, travel salesman problem, assignment problem, NP-complete, minimize, maximize, objective function, consumers, suppliers, penalties, permutations, heuristics, metaheuristics, exact algorithms, algorithm, polynomial, educational program

Poděkování

Tímto bych rád poděkoval Ing. Csc. Petru Preussovi, za podporu při psaní diplomové práce, za technické rady, připomínky a konzultace.

Prohlašuji, že diplomovou práci jsem vypracoval sám, z uvedené technické literatury jsem citoval, dále prohlašuji, že veškerý software, použitý při řešení této práce, je legální.

V Plzni

…….

Podpis

Obsah

Seznam symbolů a zkratek

A,B,C,D,E	označení bodů, vrcholů, měst
$D_1,\ldots,D_n$	dodavatelé
$S_1,\ldots,S_n$	spotřebitelé
$X_{11},\ldots,X_{ij}$	hodnota prvku v matici na pozici *i* a *j*
$C_{11},\ldots C_{ij}$	cenové ohodnocení na pozici *i* a *j*
$a_1,\ldots,a_i$	kapacita dodavatelů
$b_1,\ldots,b_i$	požadavky spotřebitelů
Z_{min}	minimalizační účelová funkce
Z_{max}	maximalizační účelová funkce
MM	maďarská metoda
NP	nedeterministicky polynomiální
ODP	okružní dopravní problém
TSP	travel salesman problem
RR	řádková redukce
SR	sloupcová redukce
GUI	grafické uživatelské rozhraní
MS VS2013	Microsoft Visual Studio 2013
HK	Hamiltonovská kružnice
V	vrchol
Hr	hranové ohodnocení
Val	hodnota (value)
ppm	parts per milion

Úvod

Přiřazovací problém patří do skupiny optimalizačních úloh, přesněji do kategorie distribučních úloh. Jedná se o přiřazení *m* dodavatelů, k *m* spotřebitelům, kde každý dodavatel může zásobovat každého spotřebitele a naopak, každý spotřebitel může být zásobován jakýmkoli dodavatelem. Každý dodavatel má s každým spotřebitelem zadán cenový koeficient za realizaci přepravy. Řešením přiřazovací úlohy je nalezení takové kombinace přiřazení, kde součet cenových koeficientů je minimální resp. maximální, dle typu úlohy.

Práce je rozdělena na dvě části. V úvodu teoretické části je popsána obecná problematika optimalizačních úloh, lineárního programování a distribučních úloh. Následuje kapitola zaměřující se na jednostupňové dopravní úlohy a jejich řešení pomocí: metody severozápadního rohu, indexové metody a Vogelovi aproximační metody. Speciálním typem těchto úloh je přiřazovací problém. Řešením tohoto problému je maďarská metoda, která je dopodrobna popsána a vysvětlena, včetně praktického příkladu.

Další část je věnována dopravnímu okružnímu problému, jeho definici, jak v oblasti matematiky, tak i v oblasti teorie grafů, výpočetní složitosti, možnostem výpočtu pomocí heuristických, metaheuristických a exaktních algoritmů. Podrobněji jsou popsány: metoda penalizací, metoda permutací a odvozená metoda vycházející z maďarské metody s eliminací parciálních smyček.

Druhá, praktická část práce je věnována programu pro řešení přiřazovacího problému maďarskou metodou a programu pro řešení okružního dopravního problému pomocí výše zmíněných metod v teoretické části. Oba programy jsou psány v jazyce C++ ve vývojovém prostředí Miscrosoft Visual Studio. U programu pro řešení okružního dopravního problému byl proveden test výkonosti, univerzálnosti a rychlosti programu pro různé velikosti vstupních zadání.

Závěr práce je věnován kompletnímu popisu uživatelského rozhraní programů a vytvoření uživatelského manuálu, pro možnost využití těchto programů k výukové činnosti.

1 Optimalizační úlohy

Podstatou optimalizační úlohy je nalezení minimální resp. maximální hodnoty účelové funkce, přičemž musí být dodrženy stanovené omezující podmínky popisující přípustné řešení úlohy. Optimalizační úlohy dělíme podle charakteru přípustných řešení na spojité a diskrétní, dále podle charakteru účelové funkce a omezujících podmínek na lineární, kombinatorické a celočíselné. Přípustné řešení diskrétních úloh se skládá z určitého počtu izolovaných bodů nebo oblastí. Nejčastěji se tento typ úloh vyskytuje v kombinatorice.

Lineární programování - Nemá nic společného s počítačovým programováním, již roku 1827, kdy byla publikována Fourier-Motzkinova eliminace se setkáváme s tímto pojmem, avšak jako zakladatele lze považovat až pana L. Kantoroviche, který roku 1939 formuloval obecné lineární programování a zabýval se jeho řešením. Lineární programování je jednou z částí výše zmíněných optimalizačních úloh, také nazývané **lineární optimalizace**.[1] Jedná se o druh matematického programování, který se skládá z účelové funkce a omezujících podmínek – tj. vlastních omezení a podmínek nezápornosti. Účelová funkce i omezující podmínky jsou vyjádřeny lineárními vztahy s konstantními koeficienty. Příklady úloh vedoucí na lineární programování jsou např. dopravní problém, distribuční problém, optimální řízení dynamických systémů, směšovací problém, optimální výrobní program a další. Mezi nejznámější metody řešení lineárního programování patří tzv. simplexová metoda (simplexový algoritmus), další známé metody jsou např. elipsoidová metoda nebo metoda vnitřních bodů.

1.1 Distribuční úlohy

Distribuční úlohy tvoří podskupinu úloh lineárního programování, mezi které řadíme jednostupňové, dvoustupňové, zobecněné, přiřazovací problémy, které mají shodný modelový typ. Díky některým specifickým vlastnostem je možné řešit tyto úlohy speciálními metodami, které nejsou tak složité jako výchozí simplexová metoda.

2 Jednostupňový dopravní problém

Jednostupňový dopravní problém je jednou z typických úloh distribučního lineárního programování. Klasickým příkladem jednostupňového dopravního problému je dopravení určitého druhu zboží, materiálů, skladových zásob, obecně zdrojů, od dodavatele k odběrateli. Je dáno *m* dodavatelů D_1, D_2, …, D_m, kde každý dodavatel má určitou kapacitu produktu a_1, a_2, …, a_m. Tento produkt je nutno rozvézt ke spotřebitelům S_1, S_2, …, S_m, kde každý vyžaduje určité množství produktu b_1, b_2, …, b_m. Dále máme definovány sazby (ceny) za přepravu c_{ij}, vyjadřující vztah mezi dodavatelem D_i a spotřebitelem S_j. V některých případech nevyjadřují c_{ij} přímo cenu, ale např. vzdálenost mezi jednotlivými spotřebiteli a dodavateli. Úkolem je zajištění všech požadavků spotřebitelů (splnění podmínek) a nalezení takového řešení, které bude odpovídat minimálním nákladům na přepravu. Zápis úlohy provádíme do tabulky.[3]

Tab. 2.1 - Vzorová tabulka dopravního problému

	Spotřebitelé				
Dodavatelé	S_1	S_2	…	S_n	Kapacita dodavatelů a_i
D_1	c_{11} x_{11}	c_{12} x_{12}	…	c_{1n} x_{1n}	a_1
D_2	c_{21} x_{21}	c_{22} x_{22}	…	c_{2n} x_{2n}	a_2
…	…	…	…	…	…
D_m	c_{m1} x_{m1}	c_{m2} x_{m2}	…	c_{mn} x_{mn}	a_m
Požadavky spotřebitelů b_j	b_1	b_2	…	b_n	

Dodavatelé jsou přiřazeni do řádek tabulky a do sloupců tabulky jsou přiřazeni spotřebitelé. V každé buňce tabulky jsou uvedeny hodnoty cenových sazeb c_{ij} a množství zboží určeného k transportu x_{ij}. V pravém sloupci uvádíme kapacitu dodavatelů a_i a v posledním řádku tabulky jsou požadavky jednotlivých spotřebitelů b_i.[3]

Matematický model jednostupňového dopravního problému

Minimalizace resp. maximalizace lineární funkce

$$\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} = Z_{min/max} \tag{2.1}$$

Obecné podmínky:

$$\sum_{j=1}^{n} x_{ij} = a_i \qquad i = 1,2,\dots,m \tag{2.2}$$

$$\sum_{i=1}^{m} x_{ij} = b_j \qquad j = 1,2,\dots,n \tag{2.3}$$

Podmínky nezápornosti:

$$x_{ij} \geq 0 \qquad j = 1,2,\dots,m;\ i = 1,2,\dots,n \tag{2.4}$$

Počet omezujících podmínek je roven $m + n$ rovnic, které mají $m \times n$ proměnných. Dopravní úlohy rozdělujeme na tři základní typy.

Vyvážená úloha – součet kapacit všech zdrojů je roven součtu požadavků všech spotřebitelů. Platí:

$$\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} bj \tag{2.5}$$

Nevyvážená úloha s přebytkem zdrojů – součet kapacit všech zdrojů je větší než součet všech požadavků spotřebitelů. Platí:

$$\sum_{i=1}^{m} a_i > \sum_{j=1}^{n} bj \tag{2.6}$$

Nevyvážená úloha s nedostatkem zdrojů – součet kapacit všech zdrojů je menší než součet všech požadavků spotřebitelů, nelze tedy uspokojit všechny potřeby spotřebitelů. [4]

$$\sum_{i=1}^{m} a_i < \sum_{j=1}^{n} bj \tag{2.7}$$

Řešení jednostupňového dopravního problému

Prvním krokem řešení úlohy je sestavení matematického modelu (4.2.1). Dalším krokem je přepsání zadání formou tabulky. Následuje kontrola vyváženosti úlohy. Není-li úloha vyvážená, přidá se fiktivní člen. Převyšuje-li kapacita dodavatelů požadavky spotřebitelů, přidáme fiktivního spotřebitele s nulovými sazbami c_{ij}. Kapacita fiktivního spotřebitele bude rovna rozdílu kapacit všech dodavatelů a všech spotřebitelů. Převyšují-li požadavky spotřebitelů, přidáme fiktivního dodavatele s nulovými sazbami c_{ij}, Kapacita fiktivního dodavatele bude rovna rozdílu kapacit všech spotřebitelů a všech dodavatelů.

Následuje nalezení výchozího řešení. Při určování výchozího řešení je nutné přiřadit hodnoty proměnných tj. kapacit dodavatelů a požadavky spotřebitelů do tabulky, tak aby řádkové součty byly rovny kapacitám dodavatelů a sloupcové součty byly rovny požadavkům spotřebitelů. Existuje několik metod nalezení výchozího základního řešení, zejména: metoda severozápadního rohu, indexová metoda, Vogelova aproximační metoda. [3]

2.1 Metoda severozápadního rohu

Postupujeme v obsazování tabulky ze severozápadního rohu tj. x_{11}. V první buňce tabulky vybereme nižší z hodnot dodavatele resp. spotřebitele a tuto hodnotu přiřadíme do buňky. Řádek resp. sloupec (dále řada), který byl vyčerpán, pomyslně vyškrtneme a vybranou hodnotu odečteme od vyšší hodnoty a dále pracujeme s tímto rozdílem. Dále se přesuneme k další buňce v opačném směru, než je vyčerpaná řada. Tímto způsobem obsadíme zbytek tabulky.

Tab. 2.2 - Metoda severozápadního rohu

	Spotřebitelé				
Dodavatelé	S_1	S_2	S_3	S_4	a_i
D_1	C_{11} **20**	C_{12} **15**	C_{13} x_{13}	C_{14} x_{14}	35
D_2	C_{21} x_{21}	C_{22} **10**	C_{23} **10**	C_{24} x_{24}	20
D_3	C_{31} x_{31}	C_{32} x_{32}	C_{33} **35**	C_{34} **20**	55
D_4	C_{41} x_{41}	C_{42} x_{42}	C_{43} x_{43}	C_{44} **5**	5
b_j	20	25	45	25	

2.2 Indexová metoda

Osazovacím kritériem této metody je hodnota sazeb c_{ij}. V tabulce postupně procházíme nejnižší hodnoty sazeb a přiřazujeme jim hodnoty kapacit dodavatelů resp. spotřebitelů, opět vždy nižší z hodnot z vybraných řad. Ostatní prvky z vyčerpané řady dále nebereme v potaz a hledáme další prvek v pořadí.

Tab. 2.3 - Indexová metoda

	Spotřebitelé				
Dodavatelé	S_1	S_2	S_3	S_4	a_i
D_1	4 **20**	8 **15**	11 x_{13}	2 x_{14}	35
D_2	23 x_{21}	14 **5**	7 **15**	3 x_{24}	20
D_3	9 x_{31}	13 x_{32}	5 **30**	1 **25**	55
D_4	35 x_{41}	18 **5**	39 x_{43}	12 x_{44}	5
b_j	20	25	45	25	

2.3 Vogelova aproximační metoda

Spočteme nejmenší diference pro každou řadu. Diference je rozdíl mezi nejmenšími cenovými koeficienty c_{ij} v řadě. Vybereme řadu s největší diferencí a v této řadě obsadíme prvek s nejnižší sazbou c_{ij}. Dojde-li ke shodě diferencí, vybereme prvek s nižší sazbou. Po osazení dále nepočítáme s vyčerpanou řadou a celý proces opakujeme.

Tab. 2.4 - Vogelova aproximační metoda

	Spotřebitelé				
Dodavatelé	S_1	S_2	S_3	S_4	a_i
D_1	4 **20**	8 x_{12}	11 x_{13}	2 **15**	35
D_2	23 x_{21}	14 **20**	7 x_{23}	3 x_{24}	20
D_3	9 x_{31}	13 **5**	5 **45**	1 **5**	55
D_4	35 x_{41}	18 x_{14}	39 x_{43}	12 **5**	5
b_j	20	25	45	25	

3 Přiřazovací problém

Přiřazovací problém je speciálním případem jednostupňové dopravní úlohy. Pro tento problém platí, že $m = n$ tj. počet dodavatelů je roven počtu spotřebitelů. Kapacity zdrojů a požadavky zákazníků se rovnají jedné a proměnná x_{ij} je bivalentní. [modely dop uloh]

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad i = 1,2,\dots,n \tag{3.1}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1,2,\dots,m \tag{3.2}$$

$$x_{ij} \in \{0;1\} \tag{3.3}$$

3.1 Maďarská metoda

Tato metoda je využívána k výpočtu dopravního přiřazovacího problému. Úloha je definována maticí sazeb c_{ij} (vzdálenost, časová náročnost, zisk, atd.). Aby bylo možné úlohu řešit maďarskou metodou (dále jen MM), musí být splněny tyto podmínky:

- Rovnost počtu řádek a sloupců (symetrická matice sazeb), pokud tato podmínka není dodržena, nutno doplnit fiktivní řadou s prohibitivními (nevýhodnými) sazbami (při minimalizaci, hodnoty řádově vyšší než nejvyšší hodnota z matice sazeb, při maximalizaci volíme 0)
- Matice sazeb musí být kvantifikovatelná.
- Kapacity dodavatelů a kapacity spotřebitelů musí být vzájemně homogenní (lze jakýmkoliv dodavatelem uspokojit jakéhokoliv spotřebitele).[4,5,11]

3.1.1 Řešení maďarskou metodou – minimalizace

Algoritmus výpočtu MM je minimalizační, požadujeme-li maximalizaci účelové funkce, je nutné převést matici sazeb na doplněk k maximálnímu prvku matice.

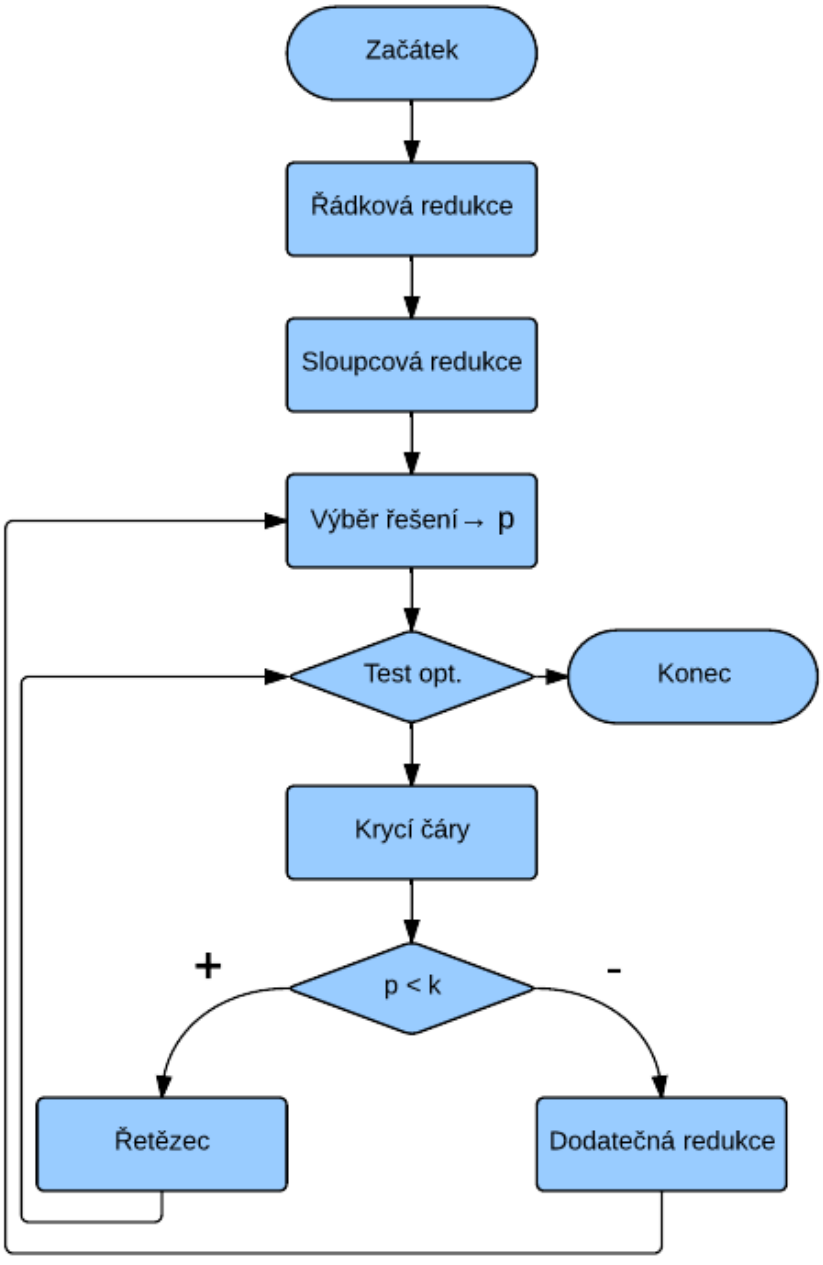
Vlastnosti algoritmu MM:

- Minimalizační proces
- Univerzální pro jakoukoliv matici
- Silně degenerovaná úloha lineárního programování

Postup řešení MM – MM je založena na hledání minimálních prvků matice, za pomocí redukce matice, tak aby každý řádek a každý sloupec obsahoval alespoň jednu nulovou hodnotu (nejmenší hodnota v řadě). Tyto hodnoty se poté zanáší do řešení a podaří-li se nalézt *n* nebo *m* nezávislých nul, jedná se o optimální řešení, pokud ne provede se dodatečná redukce a postup se opakuje.[2]

1. Zavedení **řádkové redukce** – od každého prvku v řádku odečteme hodnotu nejnižšího prvku z daného řádku.
2. Zavedení **sloupcové redukce** – ve sloupcích kde se doposud nevyskytuje nula, odečteme od každého prvku v sloupci hodnotu nejnižšího prvku z daného sloupce.
3. Provedeme výběr **nezávislých nul *p***. Nezávislá je nula, je-li vybraná jediná v řádku nebo sloupci. K výběru u rozsáhlejších matic zavádíme tzv. **incidenční indexy,** tyto koeficienty přiřazujeme nulovým prvkům a vyjadřují součet ostatních nul v řádku a sloupci dané nuly. Poté výběr nezávislých nul provádíme od nul s nejnižším incidenčním indexem.
4. Je-li počet vybraných nul roven *n* nebo *m* jedná se o **optimální řešení** a algoritmus je ukončen.
5. V opačném případě zavedeme **krycí čáry *k*.** Krycí čáry vedeme přes sloupce s vybranými nulami. Pokud zůstaly nepokryty nuly (vybrané i nevybrané) vedeme řádkové krycí čáry přes dosud nepokryté 0. Dojde-li k překrytí krycích čar na vybraných 0, zrušíme danou sloupcovou krycí čáru. Pokud došlo k odkrytí některých vybraných či nevybraných nul důsledkem zrušení sloupcových krycích čar, doplníme dodatečné řádkové krycí čáry. Tento postup opakujeme, dokud nepokryjeme všechny nuly v matici. Provedeme vyhodnocení $p < k$, pokud počet vybraných nul je menší než počet krycích čar, zavedeme řetězec.
6. **Řetězec** slouží k navýšení výběru nezávislých nul o jednu. Řetězec musí začínat a končit v řádku a sloupci, přes které nejsou vedeny krycí čáry. Nejprve vybereme nevybranou nulu, poté změníme směr o 90° a vybereme nulu vybranou, tento proces opakujeme, dokud řetězec neskončí v nepokrytém řádku nebo sloupci. Poté vyměníme vybrané nuly za nevybrané a naopak. Tímto získáme jednu vybranou nulu navíc, poté se vrátíme k bodu 4.
7. Pokud je počet krycích čar menší než *m* nebo *n* provedeme **sekundární (dodatečnou) redukci**. Vybereme nejnižší hodnotu z nepokrytých prvků a tuto

hodnotu odečteme od všech nepokrytých prvků v matici. U prvků, které jsou překryty pouze jednou krycí čarou, necháme prvky beze změny. U prvků překrytých sloupcovou i řádkovou krycí čarou, přičteme vybrané minimum. Vrátíme se k bodu 3. a tento proces opakujeme, dokud počet krycích čar není roven *m* nebo *n* [2,4,11].



Obr. 3.1 - Vývojový diagram maďarské metody

3.1.2 Příklad řešením MM

Zadána symetrická matice $n = m = 6$, s ohodnocením sazeb viz. tabulka č. 1. Úkolem je maximalizovat účelovou funkci $Z_{max.}$

64	54	51	36	39	0
33	84	41	0	1	59
48	30	25	0	13	26
78	25	16	58	70	0
58	0	22	91	22	87
3	4	61	0	51	31

Maďarská metoda je minimalizační algoritmus, tudíž je v tomto případě nutné transformovat matici sazeb. V každém řádku vyhledáme nejvyšší prvek x_{max}, nová hodnota prvku bude rovna: $x_{ij} = x_{max} - x_{ij}$.

21	31	34	49	46	85
59	8	51	92	91	33
49	67	72	97	84	71
10	63	72	30	18	88
28	86	64	5	64	1
60	59	2	63	12	32

Převedením prvků získáme tabulku č. 2, dále provedeme řádkovou redukci, od každého řádku odečteme jeho nejnižší prvek. Stejný postup aplikujeme na sloupce tabulky.

0^2	10	13	24	17	64
51	0^0	43	80	75	25
0^2	18	23	44	27	22
0^3	53	62	16	0^1	78
27	85	63	0^1	55	0^1
58	57	0^0	57	2	30

Po provedení řádkové a sloupcové redukce dostaneme tabulku č. 3. Dále zavedeme incidenční indexy ke všem nulám matice a provedeme výběr řešení. Provedeme test optimality a jelikož $\boldsymbol{p \neq n \neq m}$, nejde o optimální řešení. Jelikož $\boldsymbol{p}$ není menší než $\boldsymbol{k}$, přejdeme tedy k zavedení krycích čar.

0^2	10	13	8	17	48
51	0^0	43	64	75	9
0^2	18	23	28	27	6
0^4	53	62	0^3	0^2	62
43	101	79	0^2	71	0^1
58	57	0^0	41	2	14

U nepokrytých prvků provedeme dodatečnou redukci, vybereme nejmenší prvek a odečteme ho od všech ostatních. Prvky pokryté jednou čarou necháme být beze změny. K dvakrát překrytým prvkům přičteme výše vybraný nejmenší prvek.

0^1	10	13	6	15	46
51	0^0	43	62	73	7
0^1	18	23	26	25	4
2	55	64	0^2	0^2	62
45	103	81	0^2	71	0^1
58	57	0^1	39	0^2	12

Provedeme nový výběr řešení $\boldsymbol{p} = 5$, opět se nejedná o optimální řešení. Podmínka $\boldsymbol{p < k}$ není splněna, proto je nutné znovu vést krycí čáry a provést dodatečnou redukci.

0^1	10	9	2	11	46
51	0^0	39	58	69	3
0^2	18	19	22	21	0^2
6	59	64	0^2	0^2	62
49	107	81	0^2	71	0^2
62	61	0^1	39	0^2	12

Opět provedeme výběr řešení $\boldsymbol{p} = 5$, avšak nyní je splněna podmínka $\boldsymbol{p < k}$, proto zavedeme do řešení řetězec, po nalezení řetězce vyměníme vybrané nuly za nevybrané, tím získáme $\boldsymbol{p = m = n = 6}$ a tím optimální řešení dané úlohy $\boldsymbol{Z_{max}} = 396$.

4 Okružní dopravní problém

Okružní dopravní problém (ODP), také označovaný jako „Problém obchodního cestujícího“ (v anglické literatuře „Travel salesman problem“ - TSP) byl formulován v 19. století, na základě teorie grafů irským matematikem R. W. Hamiltonem a britským matematikem T. Kirkmanem. Jednalo se o studii kružnic u pravidelného dvanáctistěnu („dodekahedronu“), odtud pojem: **Hamiltonovská kružnice** („hamiltonovský cyklus“). HK je taková kružnice v grafu, která prochází všemi vrcholy grafu právě jednou, výjimkou je výchozí bod, z tohoto vrcholu kružnice začíná a také končí. Pokud graf obsahuje HK, nazývá se **Hamiltonovský graf**.

Aby graf mohl být označen jako hamiltonovský, musí splňovat tyto podmínky:

- Graf musí být souvislý – mezi libovolnými dvěma vrcholy existuje alespoň jedna cesta. Pokud graf není souvislý, skládá se ze souvislých částí, tyto části nazýváme komponenty souvislosti.
- Graf neobsahuje hrany typu most – typ hrany spojující komponenty souvislosti.
- Graf neobsahuje artikulace – je vrchol, který po odstranění z grafu zvýší počet komponent minimálně o jednu.
- Graf neobsahuje vrcholy stupně 1 (stupeň vrcholu udává počet hran vycházející z daného vrcholu).

Definice ODP pomocí **teorie grafů** je následující: Vstupem je úplný graf s minimálně třemi uzly (minimum pro řešitelnost úlohy s více než jedním možným výsledkem). Hranové ohodnocení nezápornými váhovými funkcemi (všem hranám e je přiřazena váha $c(e)$). Řešením úlohy je nalezení hamiltonovského cyklu, jehož suma vah všech hran grafu je minimální. Náročnost nalezení HK je **NP-úplný**. [10]

Druhým přístupem k řešení je matematický model ODP: Množina M, kde pro každé dva prvky z této množiny x a y je dána hodnota $d(x, y)$, která vyjadřuje vzdálenost x a y. Cílem je zjistit v jakém pořadí projet (odtud pojem „obchodní cestující“) všechny prvky množiny M („měst“) tak, aby každý prvek byl navštíven právě jednou a poté se vrátil do výchozího prvku a utvořil tak co nejkratší možný cyklus („cestu“). Matematicky vyjádřeno, hledáme nejmenší možný součet: [1,6]

$$d(x_1, x_2) + d(x_2, x_3) + \cdots + d(x_{n-1}, x_n) + d(x_n, x_1) \qquad (4.1)$$

4.1 Výpočetní složitost ODP

ODP patří mezi dobře strukturované úlohy s rozhodováním za jistoty, s kvantifikovatelnými proměnnými a s jediným kritériem hodnocení. Zároveň je statický, nekonfliktní a jednoetapový. Podle těchto kritérií by se mělo jednat o nejjednodušší typ úloh. Avšak složitost ODP spočívá v náročnosti výpočetní stránky. Jak bylo již výše zmíněno ODP spadají do kategorie NP – úplných úloh.

NP („Nedeterministicky polynomiální“) problém, lze řešit v polynomiálně omezeném čase na nedeterministickém Turingově stroji (počítači). Nedeterministický algoritmus, na rozdíl od algoritmu deterministického může v některých nebo ve všech krocích výpočtu vybrat libovolné z možných pokračování výpočtu. Výpočet a konečný výsledek tedy není jednoznačný z počátečního zadání a za výstup lze považovat skupinu výsledků. Výsledky lze dále procházet a určit zda alespoň některý výsledek vyhovuje zadání.

NP-úplnost (NPC, NP-complete) – skupina problémů, na kterou lze převést jakoukoliv úlohu z třídy NP. Jedná se o nejtěžší úlohy z této třídy. Neexistuje polynomiální deterministický algoritmus, který by byl schopný řešit NP-úplnou úlohu v rozumném čase. Tímto je výpočet ODP „hrubou silou“ limitován počtem prvků, jelikož jeho náročnost stoupá minimálně exponenciálně.

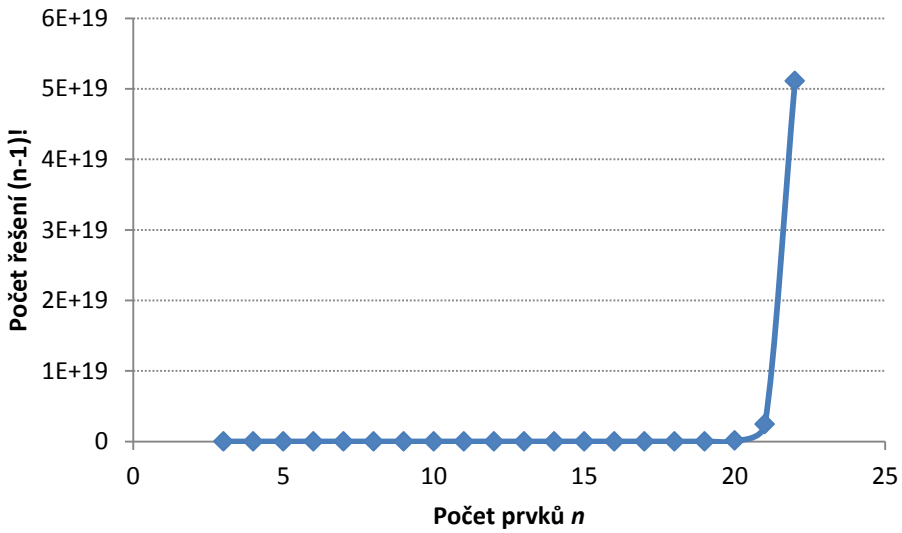
Pro lepší představu, procesor Pentium D E5300 (2,9 GHz) disponuje výpočetním výkonem 15 Gflops (float point per second). 1 flop lze přirovnat k jedné výpočetní operaci, tím dostáváme výpočetní výkon 15×10^9 operací/sec (1 operace = 0,666 ns). Algoritmy lze rozdělit na rychlé a pomalé. Mezi rychlé (polynomiální) algoritmy patří např.: n, $n\times logn$, n^2, n^3, n^4. Pomalé algoritmy se vyznačují minimálně exponenciální (nepolynomiální) závislostí, patří mezi ně např.: 2^n, n^n, $n!$. [1,6,7]

Tab. 4.1 - Časová závislost výpočtu pro množiny o n prvcích.

Operací	**n**							
	10	30	40	50	100	200	500	1000
n	0,15 µs	0,45 µs	0,60 µs	0,75 µs	1,50 µs	3,00 µs	7,50 µs	15,0 µs
n×logn	0,15 µs	0,66 µs	0,96 µs	1,27 µs	3,00 µs	6,90 µs	20,2 µs	45,0 µs
n^2	1,50 µs	13,5 µs	24 µs	37,5 µs	150 µs	600 µs	3,75 ms	15,0 ms
n^3	15,0 µs	405 µs	960 µs	1,88 ms	150 ms	0,12 s	1,88 s	15,0 s
n^4	150 µs	12,2 ms	38,4 ms	93,8 ms	1,50 s	24,0 s	15,6 min	4,16 hod
2^n	15,4 µs	16,1 s	4,58 hod	195 dní	6×10^{14} let			
Operací	**n**							
	13	15	16	17	18	19	20	21
n!	93,4 s	5,44 hod	3,63 dne	61,8 dne	3,04 let	57,6 let	1156 let	24×10^3 let

Tab. 4.2 - Počet všech možných tras ODP s n prvky

n	*(n-1)!*	*n*	*(n-1)!*
3	2	**13**	479001600
4	6	**14**	6227020800
5	24	**15**	87178291200
6	120	**16**	1307674368000
7	720	**17**	20922789888000
8	5040	**18**	355687428096000
9	40320	**19**	6402373705728000
10	362880	**20**	121645100408832000
11	3628800	**21**	2432902008176640000
12	39916800	**22**	51090942171709400000



Obr 4.1 - Grafické znázornění počtu řešení ODP s n prvky

Pozn.: Reálný počet tras je poloviční, jelikož trasa mezi třemi městy *A->B->C->A* je totožná s trasou *A->C->B->A* (stejné trasy opačný směr).

4.2 Řešení ODP

Nejjednodušším a zároveň nejvíce časově náročným výpočtem ODP je procházení všech možných cest, tj. pro *n* proměnných existuje *(n-1)!* možností. Z Tab. 4. vyplývá nemožnost řešení v rozumném čase pro vyšší počet proměnných na běžných výpočetních strojích. Ani dnešní superpočítače nejsou schopny v rozumném čase provádět výpočty pro více prvků než v řádech stovek. Vzhledem k nemožnosti použití této metody pro více prvkové úlohy a k faktu, že dodnes nebyla objevena efektivní metoda pro výpočet ODP, je nutno k řešení využít jiné alternativní metody a těmi jsou aproximační, heuristické a exaktní. Základní rozdělení aproximačních algoritmů dělíme

na **metody vytvářející řešení** a **metody řešení zlepšující.**

Metody vytvářející řešení dále dělíme na metody se **sekvenčním postupem** (tyto metody postupují lokálně, začínají v určitém bodě a z něho řešení rozšiřují, přičemž se věnují pouze okolí tohoto částečného výsledku) a na metody s **paralelním postupem** (začátek na více místech najednou, postupným rozrůstáním částečných řešení, spojení v jedno celkové). Metody sekvenční jsou obecně rychlejší a jednodušší než metody s paralelním postupem, ale vyznačují se obecně horšími výsledky.

Metody zlepšující řešení pracují s předešlým řešením získaným náhodně, nebo jednou z výše zmíněných metod a toto řešení se dále snaží vhodnými postupy vylepšit.

4.3 Heuristické algoritmy

Heuristické metody jsou metody, které dávají přípustné řešení, avšak nezaručují, zda hodnota účelové funkce bude optimální (minimální resp. maximální). Většina heuristických metod se vyznačuje vysokou odchylkou od optimálního řešení, avšak praktické výsledky jsou uspokojivé. Heuristické metody pracují v polynomiálním čase, díky tomu zvládají výpočet rozsáhlých úloh v krátkém čase. Heuristické metody nejsou exaktně formulované, a proto umožňují flexibilní úpravy pro různé varianty úloh a pro jejich specifické podmínky. Jednou z heuristických metod je metoda penalizací, které bude použita i v praktické části, proto si ji popíšeme podrobněji.

4.3.1 Metoda penalizací

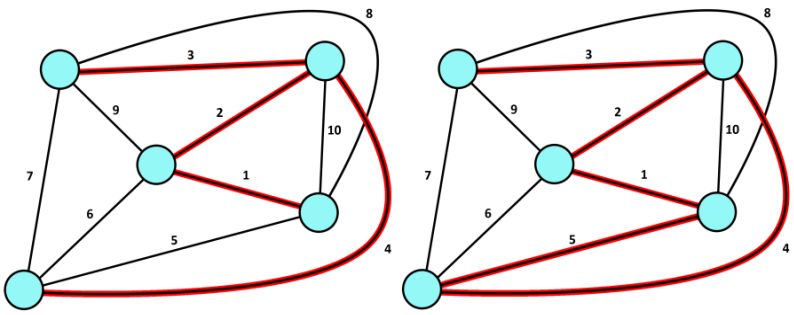
Jedná se o metodu definovanou pomocí teorie grafů. Podstatou této metody je nalezení minimálního **1-stromu** v souvislém grafu G o *n* vrcholech a *m* hranách, v případě řešení ODP se symetrickou maticí je počet hran roven (každé dva vrcholy jsou spojeny vlastní přímou hranou).

$$\sum_{i=1}^{n} n - i = m \qquad i = 1,2,\dots,n \tag{4.2}$$

Strom grafu - lze definovat jako souvislý **faktor** grafu G s libovolným stupněm vrcholů (faktor grafu G - podgraf grafu G se stejným počtem vrcholů, nikoli však hran). Faktor grafu G je stromem tehdy, má-li *n*-1 hran. Jedná-li se o hranově ohodnocený graf G, hledáme strom s minimálním součtem hranových ohodnocení. Pokud byly vybrány hrany s nejnižším ohodnocením, takto získaný strom nazýváme minimální kostra grafu.

Minimální kostru grafu - lze získat např. některým z následujících algoritmů: Kruskalův („hladový“), Borůvkův nebo Jarníkův. Do praktické části byl vybrán **Kruskalův algoritmus** (publikace 1956 J. Kruskal), tento algoritmus funguje na principu postupného připojování hran s nejmenším ohodnocením do kostry grafu, dokud nepropojí všechny vrcholy grafu. Nejprve seřadíme hrany vzestupně podle velikosti ohodnocení. Vybíráme hrany od nejmenší hodnoty a postupně přidáváme do kostry grafu, pokud by přidáním hrany vznikla v kostře grafu kružnice (smyčka), hranu nevybíráme a pokračujeme další, tento postup opakujeme, dokud bez vzniku kružnice nespojíme všechny vrcholy grafu.

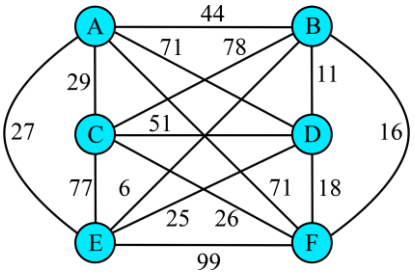
1-strom grafu – vznikne přidáním dosud nevybrané hrany s nejnižším ohodnocením do minimální kostry grafu. Nyní se rovná počet vybraných hran s celkovým počtem vrcholů. Hamiltonovská kružnice je speciálním případem 1-stromu a proto lze tvrdit, že 1-strom může být minimálním řešení dané úlohy.



Obr. 4.2 - a) Minimální kostra grafu; b) 1-strom grafu

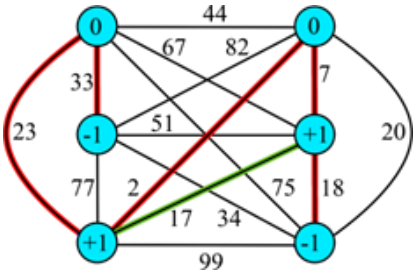
Heuristika metody penalizací – Máme graf G s hranovým ohodnocením c_{ij}, ohodnocení vrcholů grafu $t_1,\ldots,t_n$, hodnota t pro daný vrchol odpovídá stupni vrcholu (*deg(v)*). Změníme-li ohodnocení hran na $c_{ij} + t_i + t_j$, řešení ODP se nezmění, avšak mohou se změnit minimální 1-stromy. Důkazem je hamiltonovská kružnice, kde každý vrchol je druhého stupně a tudíž změna ohodnocení hran změní všechny hrany o stejnou hodnotu, tím zůstane hamiltonovská kružnice zachována, to ovšem neplatí u 1-stromů s různorodými stupni vrcholů. 1-strom není kružnicí, jestliže alespoň jeden z vrcholů není roven druhému stupni. Z toho budeme vycházet při **penalizaci vrcholů**, výchozí stav tedy zvolíme $deg(v) = 2$. Předpis penalizace vrcholu $t_i = deg(v) - 2$. Mohou nastat tři stavy, pokud stupeň vrcholu bude roven 2, hodnota penalizace bude rovna nule, vrchol nebude penalizován. Bude-li stupeň vrcholu větší než dvě, hodnota penalizace bude kladná, dojde k zvýšení ohodnocení hrany. Bude-li stupeň vrcholu 1 (menší být nemůže, vrchol by nepatřil do 1-stromu), hodnota penalizace bude záporná, dojde ke

snížení ohodnocení hrany. Nutno zvolit optimální násobek penalizace, v tomto případě vedly ke konečnému výsledku čtyři kroky, byl by násobek roven pěti, výsledek by byl dosažen třemi kroky. Obecně platí, čím vyšší násobek, tím méně kroků k nalezení řešení, avšak pouze do určité hranice. Při moc vysokém násobku může dojít k zacyklení mezi několika stavy, kde ani jeden neodpovídá řešení. Při strojovém výpočtu volíme násobek jedna, vede k nejvíce krokům, avšak k největší pravděpodobnosti nalezení řešení. Metoda penalizací je optimalizační heuristikou, která buďto nalezne přesné řešení nebo nevrátí žádný výsledek (zacyklení).[16]



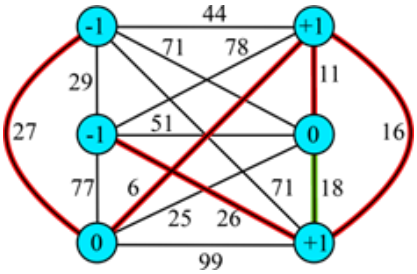
Příklad

Graf G: počet vrcholů n = 6, počet hran m = 15, s hranovým ohodnocení c_{ij}, penalizace vrcholů zvolena $t_i = 4 \times (\deg(v_i)-2)$.

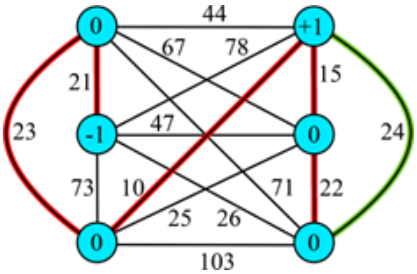


Určení **minimální kostry grafu** – označena červeně, určení **1-stromu** – označen červeně + zeleně.

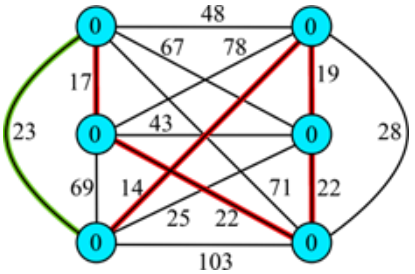
Jelikož 1-strom nesplňuje kritéria hamiltonovské kružnice, nejedná se tedy o řešení ODP a je nutno provést **penalizaci** vrcholů grafu. Upravíme ohodnocení všech hran grafu $c_{ij} = c_{ij} + t_i + t_j$.



Provedeme nový výběr minimální kostry grafu a 1-stromu. Tento proces opakujeme, dokud nebudou splněna kritéria hamiltonovské kružnice.



Řešením úlohy ODP metodou penalizací je hamiltonovská kružnice: *A->C->F->D->B->E->A*



4.3.2 ODP odvozenou maďarskou metodou

Další heuristikou je metoda odvozená z maďarské metody, která je určena pro řešení přiřazovacího problému. Tyto metody jsou si v základu velice podobné. Pokud by se použilo obecné maďarské metody ODP, dostali bychom řešení, ale mohly by nastat tyto nežádoucí stavy:

- Mohl by být vybrán prvek na diagonále, cesta z bodu *X* do bodu *X*.
- Vybrané prvky maďarskou metodou vracejí vždy optimální řešení (přiřazení), ovšem takovýto výběr ve většině případů netvoří jednu okružní cestu ale několik smyček.

První bod je snadno eliminovatelný, např. vhodným zvolením velikosti prohibitivních hodnot na diagonále nebo obecnému zákazu vybírání prvků na diagonále. Avšak ošetření druhého bodu již není tak jednoduché. Jedno z odvození této metody popisuje pan Švasta, někdy označováno jako Švastova metoda, jiné odvození v cizí literatuře označováno pouze jako řešení pomocí maďarské metody bylo použito zde.

Heuristika ODP odvozenou MM - Z maďarské metody je provedena sloupcová a řádková redukce, následuje výběr řešení, nejedná-li se o optimální řešení, pokračujeme dodatečnou redukcí resp. řetězce, dle vyhodnocení podmínky $p < k$. Proces opakujeme, dokud není možné provést výběr optimálního řešení.

Od této části se algoritmy liší. Pokud by výběr pomocí maďarské metody tvořil optimální okružní cestu (hamiltonovskou kružnici), jednalo by se i o optimální řešení ODP, ovšem takových případů je minimum. Proto je nutno upravit algoritmus, aby vybíral co nejmenší optimální řešení, avšak byla splněna podmínka okružní cesty.

Vycházíme tedy z optimálního řešení maďarské metody, avšak pokud řešení netvoří okružní cestu, zrušíme výběr a v matici najdeme nejmenší nenulový prvek (jedná se variaci dodatečné redukce), který zavedeme do řešení. Zbytek výběru postupuje podle pravidel maďarské metody, pokud ani tento nový výběr nesplňuje podmínku okružní cesty, přejdeme k dalšímu prvku v pořadí. Postup opakujeme, dokud nenalezneme řešení splňující podmínku okružní cesty. Pokud by nastal stav, že nebylo nalezeno žádné řešení, provedeme výběr nejmenšího prvku a k němu vybereme další v pořadí, zbytek výběru probíhá klasicky maďarskou metodou. Proces opět opakujeme, dokud nebude nalezeno optimální řešení. Tímto postupem vždy dostaneme řešení splňující podmínku okružní cesty. Zdali se jedná o optimální řešení úlohy nelze stanovit, jelikož tato metoda spadá do skupiny heuristických algoritmů.[17,18]

4.4 Metaheuristické algoritmy

Heuristické algoritmy bývají definovány pro konkrétní úlohu, avšak některé heuristické metody zobecnit a použít k řešení obecných optimalizačních úloh. Tyto obecné metody nazýváme Metaheuristiky. Funkce těchto metod spočívá v zapojení více heuristik k řešení jedné úlohy a tímto nabývají na efektivnosti. Jedná se o algoritmy postupně zlepšující řešení. Mezi tyto algoritmy patří např.: **metoda lokálního hledání** (local search), z ní vycházející **metoda zakázaného prohledávání** (tabu search), **horolezecký algoritmus** (hill climbing), od něj odvozená metoda **simulovaného žíhání, elastické neuronové sítě,** či tzv. **genetické algoritmy**.

Genetické algoritmy aplikují principy živých organizmů. Na jednotlivé prvky množiny pohlížíme jako na živé organizmy v cizím prostředí, které se snaží přežít. Úspěšnost adaptace a reprodukce nám udává, o jak úspěšné řešení se jedná. Hledání optimálního výsledku pak probíhá ve výběru počátečních prvků (populace), provedení simulace vývoje s evolučními mechanizmy (dědičnost, mutace, reprodukce, přirozený výběr atd.) a následné vyhodnocení, chybná řešení mají tendenci vymírat naopak správná přežívat a dále se reprodukovat.

4.5 Exaktní algoritmy

Exaktní algoritmy vycházející z lineárního programování vždy naleznou optimální řešení, za předpokladu, že řešení existuje. Jejich nevýhodou jak již bylo výše zmíněno, je exponenciální nárůst složitosti s narůstajícím počtem prvků. Mezi nejpoužívanější v této kategorii patří např.: kombinatorická **metoda větvení a mezí** (branch and bound). Tato metoda byla poprvé publikována v roce 1960 (A. H. Land a A. G. Doig). Jedná se o nepolynomiální algoritmus založený na prohledávání kořenového stromu do hloubky nebo do šířky. Rozděluje množiny (větve) možných řešení a hledá v nich optimální přiřazení. Dalším algoritmem je **metoda řezných (sečných) nadrovin** (cutting-plane)**.** Tato metoda je vhodná pro celočíselné a smíšeně celočíselné úlohy, není vhodná pro řešení bivalentních (hodnoty pouze 0 a 1) úloh. Úlohu řešíme zanedbáním podmínek celočíselnosti, pokračujeme simplexovou metodou, pokud výsledné řešení splňuje podmínky celočíselnosti, výsledek je optimální, pokud ne přidáme lineární omezení a dále opět pokračujeme simplexovou metodou, postup opakujeme, dokud nezískáme optimální řešení. Dále lze zmínit metodu **větví a řezů** (branch and cut), jedná se o kombinaci výše zmíněných algoritmů.[1,2]

4.5.1 ODP Permutace

Jednou možností jak získat řešení ODP je projít všechna možná řešení úlohy, např. pomocí permutací. Jak již bylo zmíněno výše, jedná se o typ úloh s exponenciálně narůstajícím počtem řešení při lineárním zvyšování velikosti zadání. Proto nelze hovořit o této metodě jako efektivní, avšak pro naše účely poslouží. Jak lze vidět v tabulce 5., lze na běžném osobním počítači pracovat s 13 prvky (městy), s výpočetní rychlostí v řádu jednotek minut. Počet řešení úlohy ODP pomocí permutací získáme předpisem **(*n*-1)!/2**.[1,6]

Vývoj programu:

Metoda je řešena pomocí práce s řetězci, kdy je zadána posloupnost prvků v jednom řetězci (např. ABCDE) a ten je postupně rozdělována na podřetězce a postupně zase skládán, opět je zde využito rekurze. Výsledná metoda vypadá následovně:

```
Void permutation(String^ prefix, String^ str, String^ kons) {
int n = str->Length, zFun;
if (n == 0){
   if (prefix->Substring(0, 1) != kons)return;
       for (int i = 0; i < X-1; i++){
          zFun += Int32::Parse(textBoxes[Int32::Parse(prefix->Substring(i, 1)),
          Int32::Parse(prefix->Substring(i + 1, 1))]->Text);
       }
       zFun += Int32::Parse(textBoxes[Int32::Parse(prefix->Substring(X-1, 1)),
       Int32::Parse(prefix->Substring(0, 1))]->Text);
       vsechnyMoznosti[faktPocet] = gcnew permut;
       vsechnyMoznosti[faktPocet]->value = zFun;
       vsechnyMoznosti[faktPocet]->z = prefix;
       faktPocet++;
       zFun = 0;
   }
   else {
       for (int i = 0; i < n; i++){
           permutation(prefix + str[i], str->Substring(0, i) + str->Substring(i +
           1, str->Length - (i+1)),kons);
       }
   }
}
```

5 Praktická část – Výukový program

5.1 Maďarská metoda

Úvod praktické části (programu) se zabývá řešení přiřazovací úlohy pomocí maďarské metody.

Vývoj programu:

Praktická část byla zpracovávána ve vývojovém prostředí Microsoft Visual Studio 2013 for desktop (dále jen VS) a psána v programovacím jazyce C++. Prvními kroky bylo seznámení se prostředím VS, jde o multiplatformní vývojové prostředí. V posledních let se v syntaxi jazyka C++ lecos změnilo, například místo tečkové notace se v VS používají šipky („->”). Dalším úskalím bylo zpracování tohoto projektu s grafickým rozhraním. Pro praktickou část diplomové práce není vhodný výstup do konzole. VS disponuje velice intuitivním průvodcem pro vývoj grafického rozhraní, díky kterému bylo možno navrhnout základní GUI. Bohužel vzápětí se vyskytl problém dynamického umisťování prvků (tlačítek, textových polí, labelů, atd.) nebo průběžné změny rozlišení plochy. S tímto si bohužel průvodce GUI nedokázal poradit a při prvním užití dynamického prvku nebyl průvodce GUI nadále přístupný a vše muselo být programováno ručně.

Základní GUI tedy obsahovalo tlačítka Solve, Matrix a Random, dále vstupní textové pole a label (popisek) pro účelovou funkci. Základní funkce programu tedy byla vygenerování textových polí ve tvaru symetrické matice o velikosti zadané v úvodním textovém poli při stisknutí tlačítka Matrix.

Hodnoty z matice jsou ukládány do dvourozměrného pole *values[i, j],* které si lze snadno představit jako samotnou matici, kde hodnoty jsou indexovány pomocí souřadnic *i* a *j.*

Vývoj programu důsledně sledoval algoritmus vývojového diagramu maďarské metody. První byly funkce pro řádkovou a sloupcovou redukci, kde výstup byl řešen výpisem do labelů pod generovanou matici, toto řešení bylo nakonec ponecháno jako finální výstup celého programu. Jednalo se o seřazení prvků od nejmenšího po největší a odečtení nejmenšího prvku od sebe sama a všech ostatních.

Následující funkce je o něco náročnější, šlo o výběr řešení. Skládala se z několika částí, první bylo přiřazení incidenčních indexů k nulám v redukované matici. Toto bylo řešeno procházejícím algoritmem dvourozměrného pole pomocí dvou cyklů

„for“, tímto bylo možné procházet tabulku řádku po řádce. Vyhledávající kritérium byla nula, vždy když v matici byla nalezena nula, bylo nutno uložit její pozici tedy indexy *i* a *j*. Poté dvakrát procházet matici jednou sloupec a jednou řádek, vždy s jednou neměnnou souřadnicí a vyhledat všechny nuly, nakonec odečíst hodnotu dva, jinak by sama nalezená nula byla započtena jako incidenční jednou ze sloupce, jednou z řádky.

Následoval výběr řešení, nuly byly vybírány podle velikosti incidenčního indexu, v případě více prvků se stejnou hodnotou bylo postupováno metodou severozápadního rohu. Do výběru musela být zanesena podmínka, zda již v řádku nebo sloupci nebyla vybrána jiná nula. To bylo řešeno obarvením labelu vybrané nuly, v tomto stádiu programu ještě nebyly užity struktury, díky kterým můžeme prvku na dané pozici přiřadit více vlastností (např.: int hodnotu, boolean true/false (vybraná/nevybraná), int incidenční index).

Po výběru řešení následovala funkce testu optimality, jednoduchá procházející funkce hledající všechny obarvené prvky. Podle výstupu této funkce, byl algoritmus buďto ukončen nebo bylo nutné přejít k zavedením krycích čar.

Funkce krycích čar, byla realizována procházením matice ne po řádkách ale po sloupcích a při každé nalezené vybrané nule byla do pole pro krycí čáry (*coverX[počet sloupců]* a *coverY[počet řádků]*) přiřazena hodnota *true*. Tímto byly získány výchozí sloupcové krycí čáry. Po tomto primárním výběru byl do nekonečného cyklu (do{} while;) dán vyhledávající algoritmus pro doposud nevybrané nuly a při nalezení byla zaevidována daná řádková krycí čára a v této řadě byly nalezeny ostatní vybrané nuly a jejich sloupcové krycí byly nastaveny na hodnotu false. Algoritmus skončil poté, co neproběhlo žádné další možné připsání řádkové krycí čáry.

Poté co byla tato funkce vyhodnocena podmínkou $p < k$, přešel program do části dodatečné redukce nebo řetězce. Při dodatečné redukci bylo využito krycích čar, v matici byly vyhledány prvky bez krycích čar, seřazeny a nejmenší prvek byl uložen do pomocné proměnné. Každý prvek v matici byl upraven podle definice dodatečné redukce. Byl-li nepokryt žádnou krycí čarou, byl od něj nejmenší prvek odečten. Pokud byl prvek pokryt jednou krycí čarou, prvek se nechal být. A k dvakrát překrytým prvkům se nejmenší prvek přičetl. Poté byl algoritmus přes příkaz *goto* poslán zpět k výběru řešení, kde dál pokračoval.

Nejtěžší částí programu bylo programování vyhledávání řetězce. Přesný postup zde bude pouze nastíněn. Základní princip řetězce je procházení matice dokud nenalezneme nevybranou nulu v řádku či sloupci, který doposud neobsahuje vybranou

nulu, a od ní pak zahájíme hledání nuly vybrané v kolmém směru k původnímu nevybranému sloupci resp. řádku, v kterém se nachází, při každém nalezení opačné nuly, než na které se právě nacházíme, mění řetězec směr o devadesát stupňů. Samotné nalezení jednoho řetězce nebyl problém, avšak aby byl algoritmus schopen vrátit se pokaždé, kdy nenalezl platný řetězec o jeden krok zpět a zkusil prohledat zbytek řádku resp. sloupce, program výrazně koplikovalo. Nakonec řešení spočívalo v rekurzivní funkci (funkce, která volá sama sebe). Část výsledné metody vypadá takto:

```
Void findNext(int z, int y, int x, Boolean statSwiche, Boolean ifSel){
   Boolean switcher = statSwiche, selUnsel = ifSel;
   if (values[y, x] == 0 && checkYelY[y] == true) switcher = true;
   if (values[y, x] == 0 && checkYelX[x] == true) switcher = false;
   chain[z, y, x] = 1;
   if (switcher == false){
        for (int j = 0; j < X; j++){
/*začátek bloku*/
if (values[y,j] == 0 && selUnsel == true && textLabels[y, j]->BackColor == Yellow){
                chain[z, y, j] = 1;
                switcher = !switcher;
                selUnsel = !selUnsel;
                findNext(z, y, j, switcher, selUnsel);
                switcher = !switcher;
                selUnsel = !selUnsel;
                if (konec == true)return;} /*konec bloku*/
...}
```

Celá metoda se pak skládá ze čtyř bloků, dva pro vybrané resp. nevybrané nuly a dva pro kolmý resp. vodorovný směr. Po provedení řetězce je algoritmus opět vrácen k testu optimality a pokračuje dále.

Tímto je kód algoritmu maďarské metody u konce, alespoň jeho primární funkce, kód obsahuje i další části, které ovšem nejsou klíčové a proto zde nejsou uvedeny. Finální výstup programu lze vidět na *Obr. 5.5*.[13,14,15]

Popis rozhraní:

- **Solve**: Tlačítko vyvolávající řešení programu.
- **Matrix**: Generace matice o velikosti zadané hodnoty do TextBoxu nad tlačítkem.
- **Random**: Generace náhodný čísel (0-99) do vygenerované matice (nejnáročnější při vývoji programu bylo nekonečné, ruční zadávání čísel do matice).
- **Z=**: Výsledek účelové minimalizační funkce.
- **Tabulka TextBoxů**: Vstupní tabulka vygenerovaná pomocí tlačítka Matrix.
- **Tabulka Labelů**: Výstupní tabulka labelů, s průběžným a finálním řešením úlohy.
- **Průběh řešení**: Postupný výpis kroků řešení.

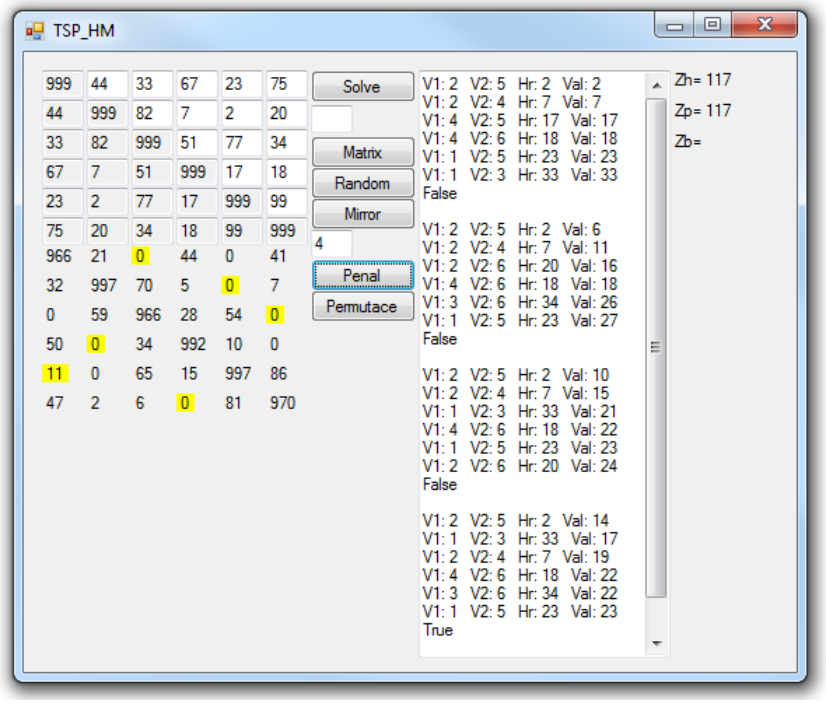
5.2 Metoda penalizací

Tuto metodu jsem zpracoval v praktické části, jako součást výsledného programu. Algoritmus programu postupuje dle výše zmíněných postupů.

- Načtení a seřazení všech hodnot tabulky (hran grafu), k řazení použit algoritmus bubble sort.
- Nalezení minimální kostry grafu – nejobtížnější část algoritmu, nakonec řešeno pomocí rekurze (funkce, která volá sama sebe):

```
Boolean findSkelet(int k, int start, int konec){
for (int i = 0; i < countEdge; i++){
if ((hrany[i]->v1 == konec || hrany[i]->v2 == konec) && hrany[i]->select
== true && (i != k)) {
if ((hrany[i]->v1 == start && hrany[i]->v2 == konec) || (hrany[i]->v2 ==
start && hrany[i]->v1 == konec)){ fail = false; return false; }
if (hrany[i]->v1 == konec){ findSkelet(i, start, hrany[i]->v2); }
if (hrany[i]->v2 == konec){ findSkelet(i, start, hrany[i]->v1); }}}
return fail;}
```

- Následuje doplnění na 1- strom
- Určení stupňů řad (vrcholů grafu)
- Vyhodnocení výsledku, pokud všechny stupně = 2, vrácení TRUE (konečné řešení + výpis řešení). V opačném případě návratová hodnota FALSE a nutné opakování celého algoritmu.



Obr. 5.1 - Výstup programu pro řešení ODP

Výstup programu: textový výpis 1-stromu z každého kroku. *V1* – vrchol 1, *V2* – vrchol 2, *Hr* – ohodnocení hrany zadané matice, *Val* – hodnota ohodnocení hrany.

5.3 ODP odvozenou maďarskou metodou

Další a poslední částí programu je řešení ODP pomocí odvozené MM. Tímto je splněn třetí bod zadání, navrhnutí metody vyloučení parciálních smyček v okružním dopravním problému, avšak pouze pro obor přípustných řešení. Metoda pracuje v polynomiálním čase, avšak jako ostatní tyto metody neposkytuje jistotu optimálního řešení.

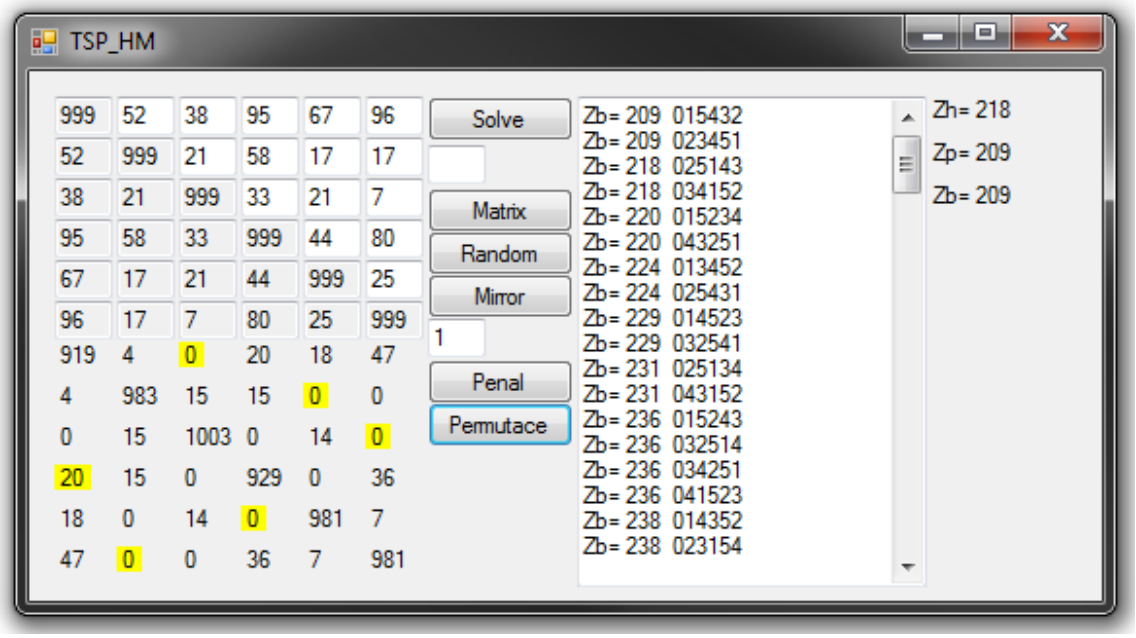
Vývoj programu:

První část kódu byla použita z maďarské metody, v GUI byla změněna zadávací matice, která byla omezena pouze na prvky nad diagonálou. ODP úloha se stejnou vzdáleností mezi $A \rightarrow B$ a $B \rightarrow A$ má symetrickou matici. Diagonálu osadíme prohibitivními hodnotami, nelze jet z místa A → A. Přidáno tlačítko **Mirror**, které nám zrcadlově nakopíruje horní část matice pod diagonálu. Dále je tato metoda spojena s metodou penalizací a výpočtem hrubou silou pomocí permutace. Obě tyto metody vracejí přesné optimální řešení (metoda penalizací pouze pokud nalezne řešení) na rozdíl od odvozené MM metody, jsou zde zavedeny pro kontrolu výsledků a pro odhad chyby této metody. Metoda penalizací pracuje v polynomiálním čase. Metoda permutací má exponenciální nárůst řešení, viz tabulka 5. v kapitole 4.1. Proto tuto metodu lze použít pouze matice do velikosti 8×8 a i zde je výpočet poměrně dlouhý. Pro řešení těchto metod slouží tlačítka **Penal** a **Permutace.**

Hlavním rozdílem od původní maďarské metody bylo vyřešení okružní cesty (hamiltonovské kružnice). Pro tento účel byla napsána funkce *ifCircle()*, která ověřuje, zda vybrané řešení je nebo není okružní cestou. Její funkce je založena na postupném procházení prvků z výběru a kontrole, zda není vytvořena kratší smyčka, než je požadovaná velikost matice, návratová hodnota funkce je *true/false*. Při vrácení hodnoty *false* bylo nutné přistoupit k výběru nenulového prvků do řešení. K tomuto slouží funkce *doCircle(),* která je následně opět ověřena předchozí funkcí, dokud není nalezeno optimální řešení. [13,14,15]

Popis rozhraní:

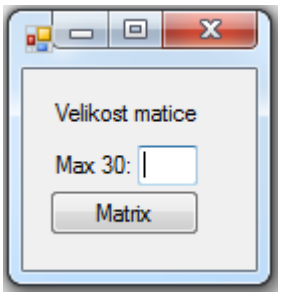
- **Solve**: Tlačítko vyvolávající řešení programu.
- **Matrix**: Generace matice o velikosti zadané hodnoty do TextBoxu nad tlačítkem, s možností zadání pouze od diagonály výše.
- **Random**: Generace náhodný čísel (0-99) do vygenerované matice, zrcadlově symetrické.
- **Mirror**: Zrcadlové doplnění tabulky pod diagonálu.
- **Penal**: Tlačítko pro provedení a výpis penalizační metody.
- **Permutace**: Tlačítko pro provedení a výpis výpočtu pomocí permutace.
- **Výstup TextBoxu**: Výstup pro metody penalizací a výpočtu permutací.
- **Tabulka TextBoxů**: Vstupní tabulka vygenerovaná pomocí tlačítka Matrix.
- **Tabulka Labelů**: Výstupní tabulka labelů, s průběžným a finálním řešením úlohy.
- **Z_h=**: Výsledek účelové minimalizační funkce pro odvozenou maďarskou metodu.
- **Z_p=**: Výsledek účelové minimalizační funkce pro metodu penalizací.
- **Z_b=**: Výsledek účelové minimalizační funkce pro výpočet pomocí permutace.



Obr. 5.2 - Výstup programu pro metodu odvozené MM a pro výpočet permutací.

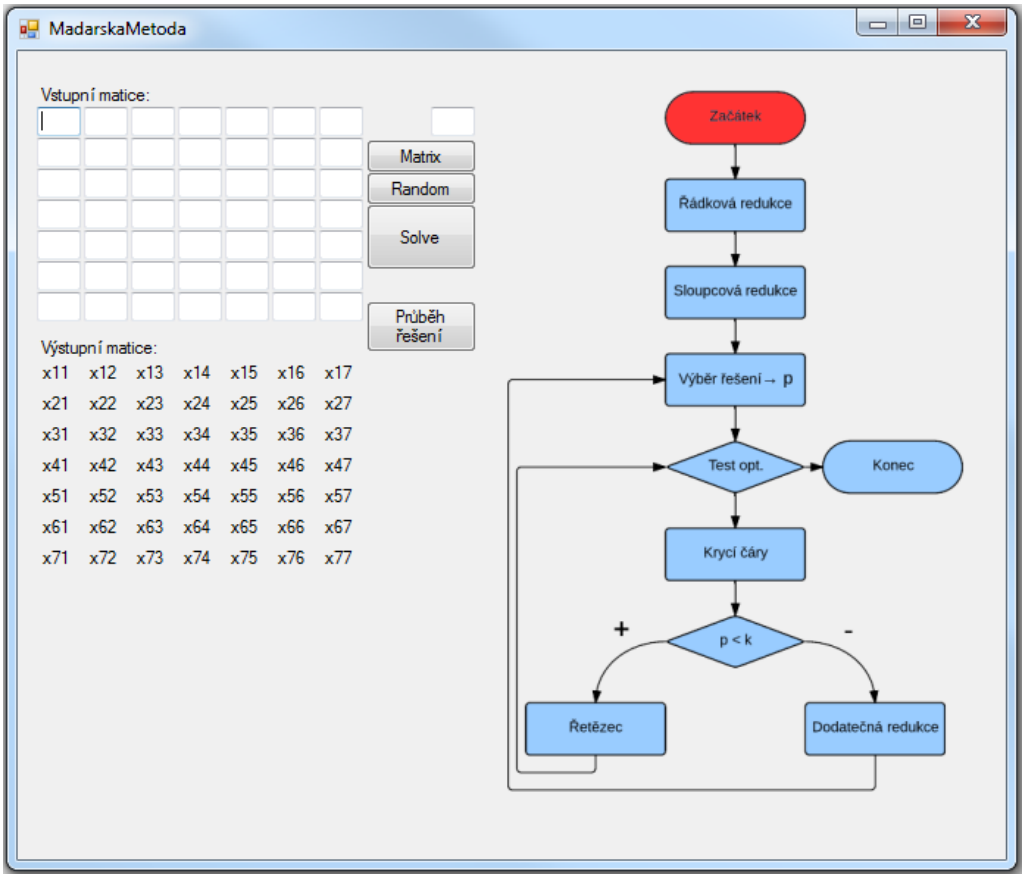
5.4 Uživatelský manuál – Maďarská metoda

Ke spuštění je nutná instalace knihoven Microsoft Visual Studia, ve kterém byl program napsán. Ze složky programu nutno nainstalovat vcredist_x86.exe, obsahující výše zmíněné knihovny. Po instalaci, bude možno spustit výsledný program na jakékoli stanici s operačním systémem Windows x86/x64.



Obr. 5.3 – Úvodní spuštění programu

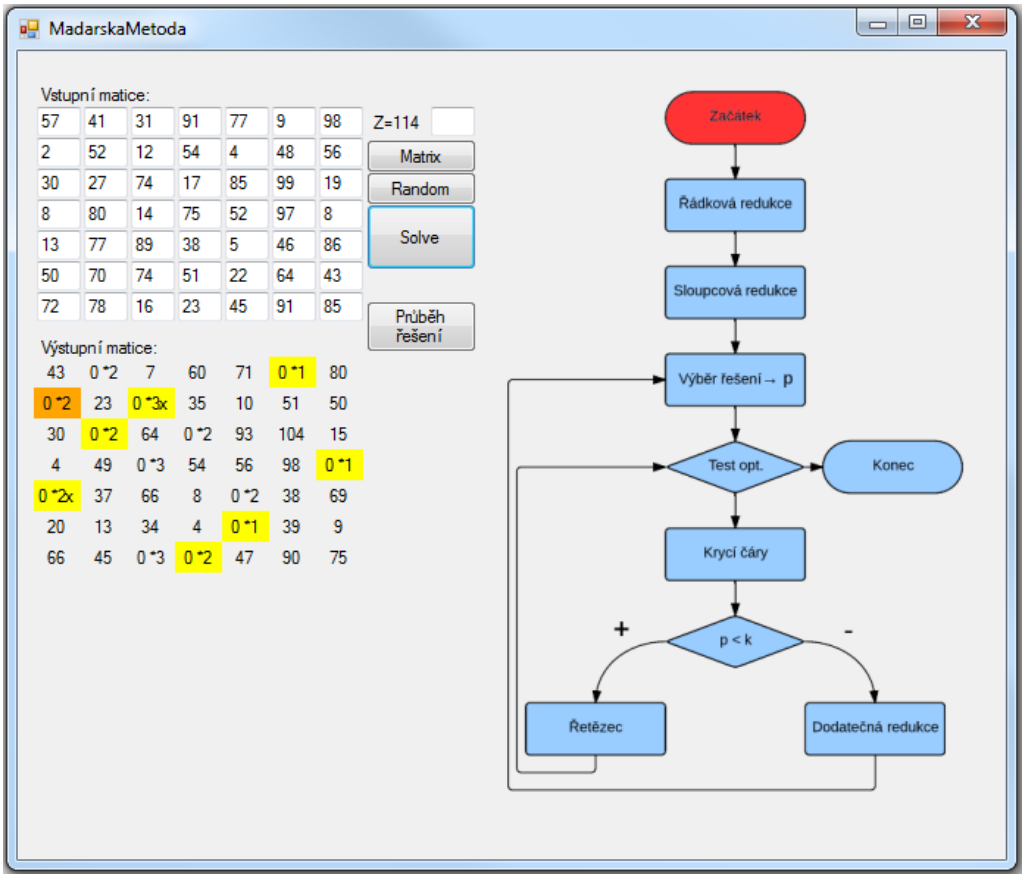
Po spuštění programu MMD_VS13.exe (MaďarskáMetoDa_VisualStudio2013), bude vygenerováno okno s výzvou zadání velikosti matice jak je vidět na *Obr. 5.4*, zadává se pouze jedno číslo, k vygenerování čtvercové, souměrné matice o daném rozměru.



Obr. 5.4 - Vygenerovaná pracovní plocha s maticemi 7x7

Po potvrzení tlačítka Matrix dojde k vygenerování pracovní plochy programu *Obr. 5.4*. V levé části se nachází vstupní matice textových polí, pod ní výstupní matice polí popisků. Textové pole nad tlačítkem Matrix, slouží k přegenerování velikosti matic. Pomocí tlačítka Random lze generovat matici náhodných čísel s hodnotami 0-99, program je omezen na dvouciferná čísla.

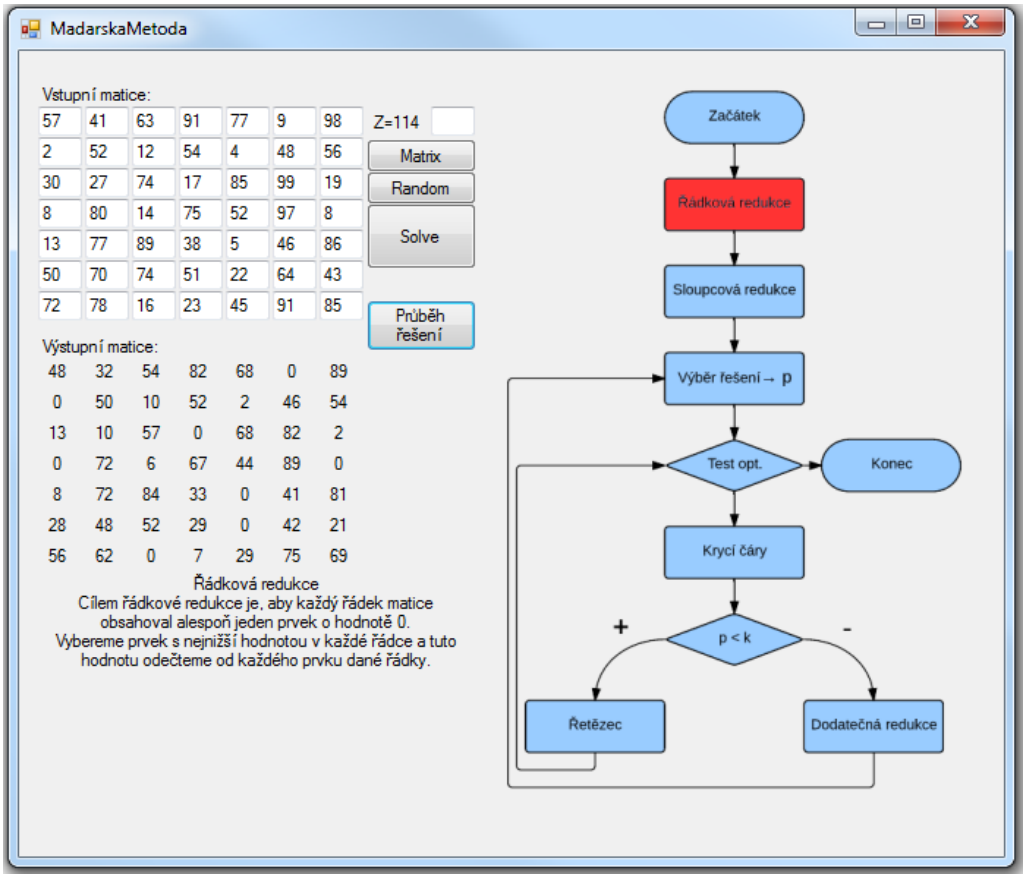
Po vyplnění vstupní matice, buď možností tlačítka Random, nebo ručně lze spustit řešení daného zadání tlačítkem Solve *Obr. 5.5*. Výsledek bude vypsán do výstupní matice, hodnota prvků bude odpovídat změnám v průběhu řešení, nikoli hodnotám ze zadání.



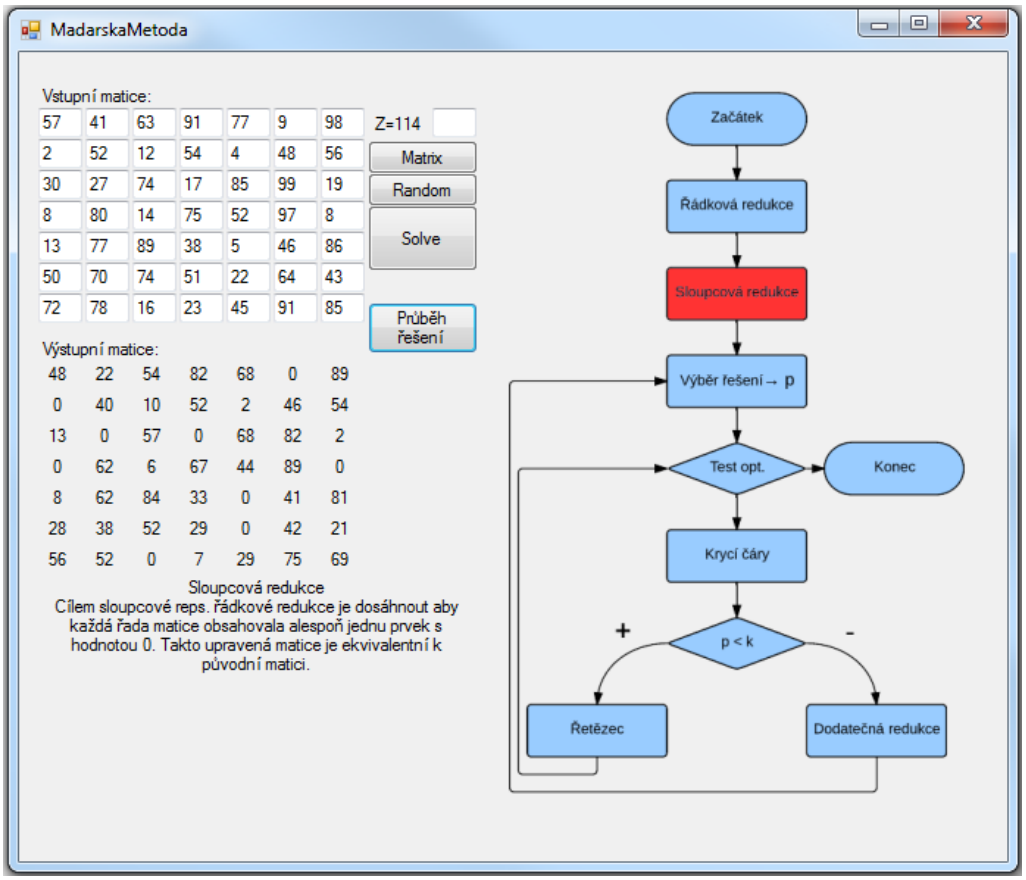
Obr. 5.5 – Výstup, základní optimální řešení

Žluté jsou prvky základního optimálního řešení, oranžové prvky jsou prvky, které byly vybrány před krokem „řetězec“, za tyto prvky byly nahrazeny kolmo ležící žluté prvky, dle pravidel řetězce.

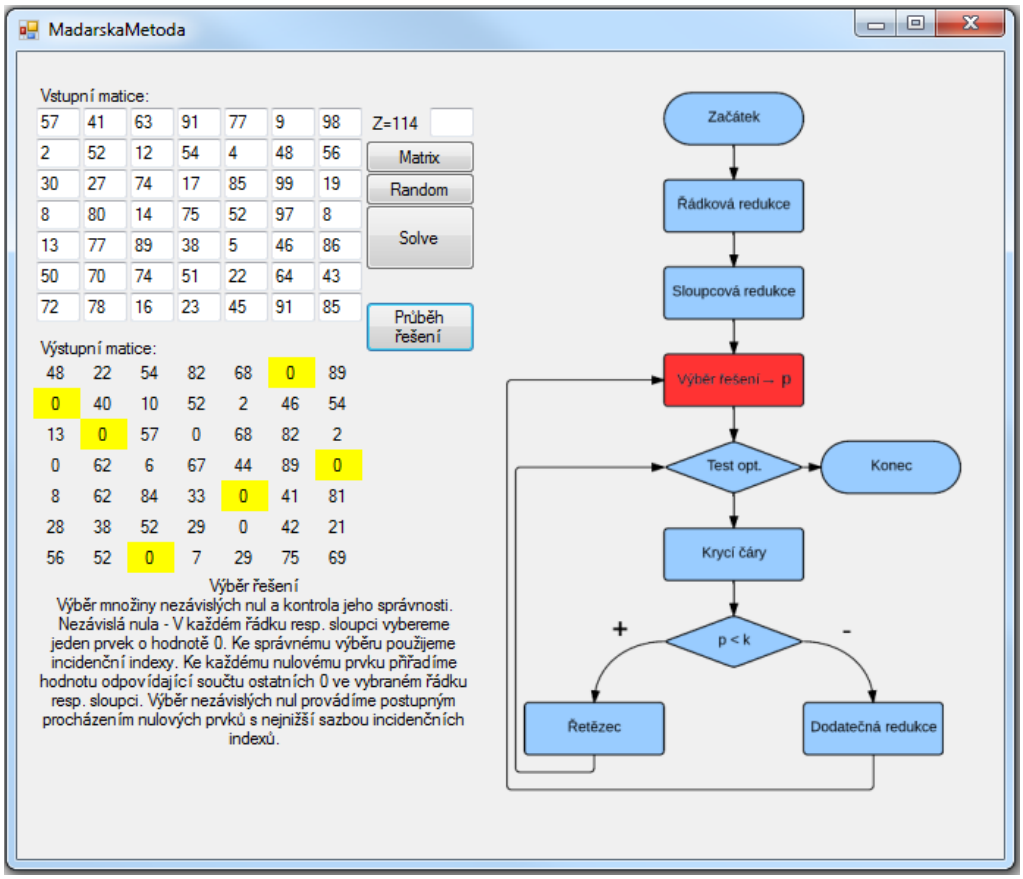
Poslední částí je zobrazení průběhu řešení pomocí stejnojmenného tlačítka. S každým stiskem tohoto tlačítka je proveden jeden krok výpočtu, jaký lze vidět na vývojovém diagramu v pravé části programu Obr. 5.6 – 5.13.



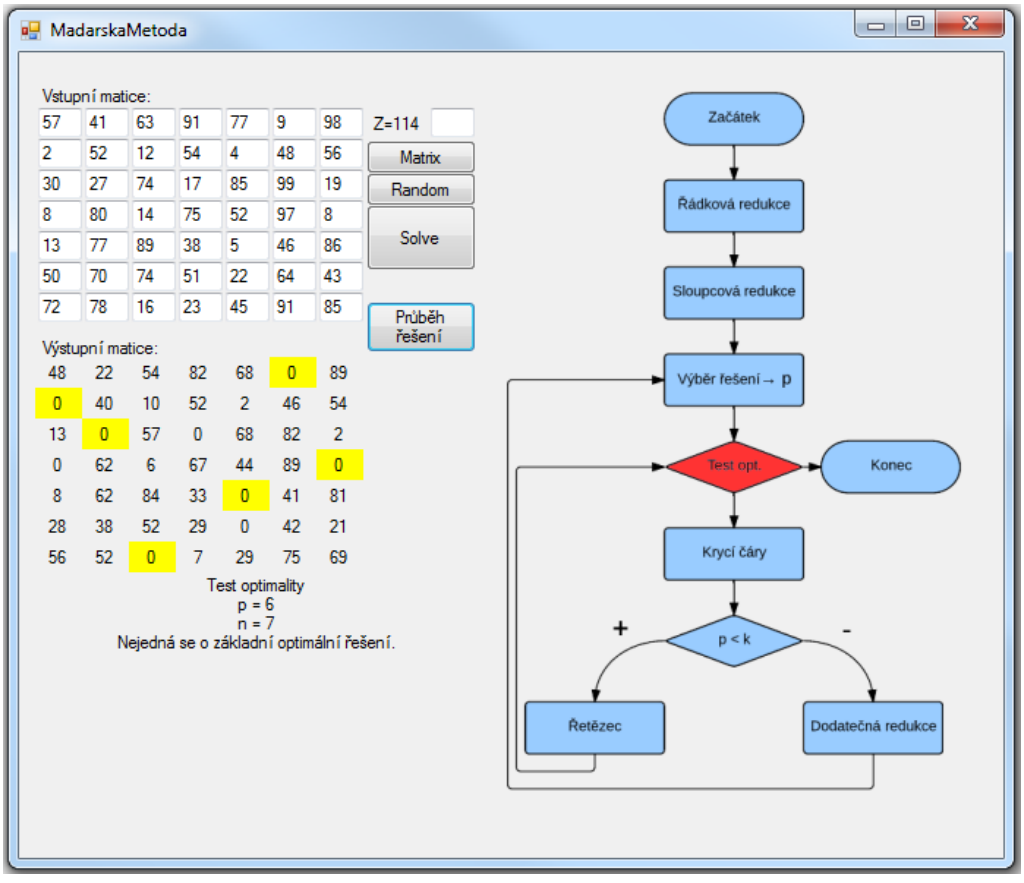
Obr. 5.6 – Výstup programu – Řádková redukce



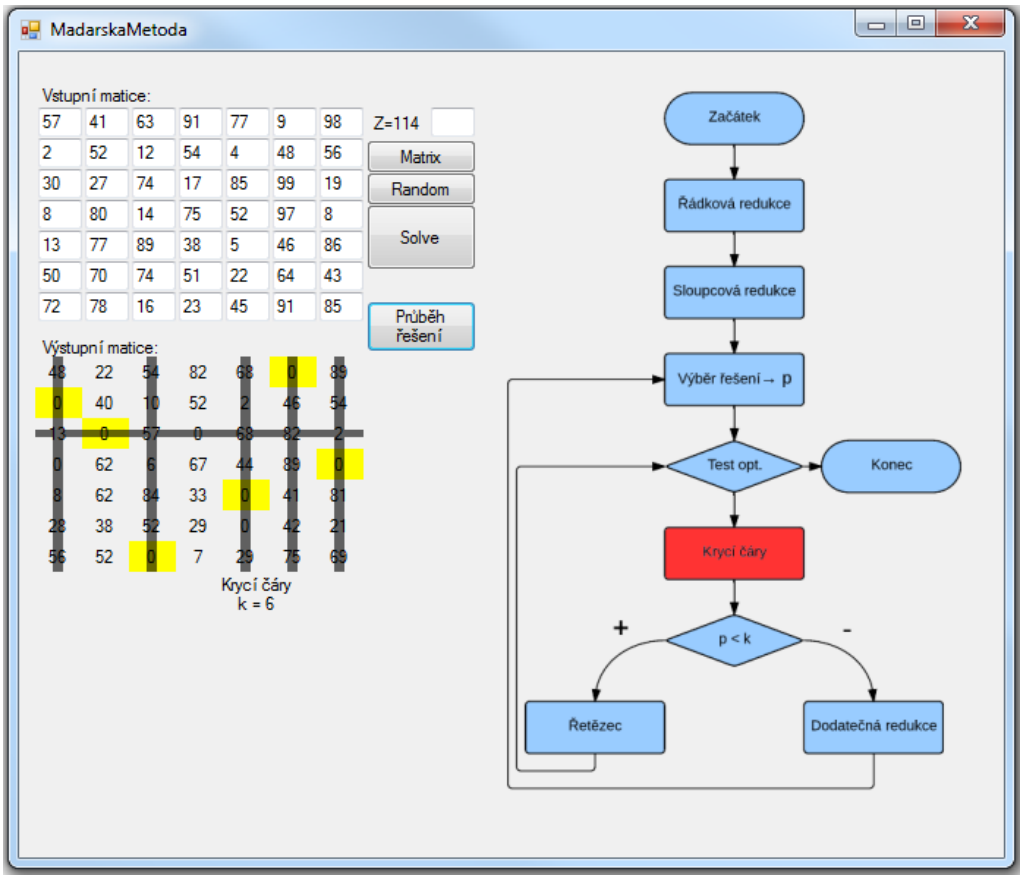
Obr. 5.7 – Výstup programu – Sloupcová redukce



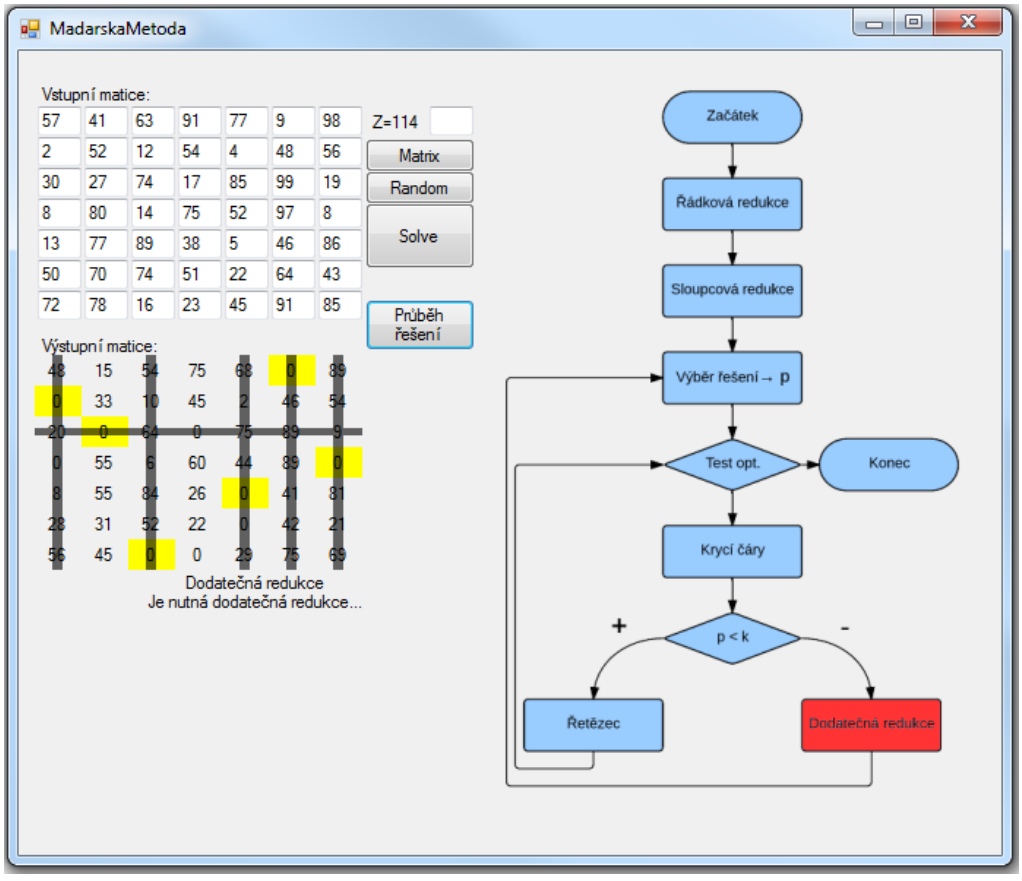
Obr. 5.8 – Výstup programu – Výběr řešení



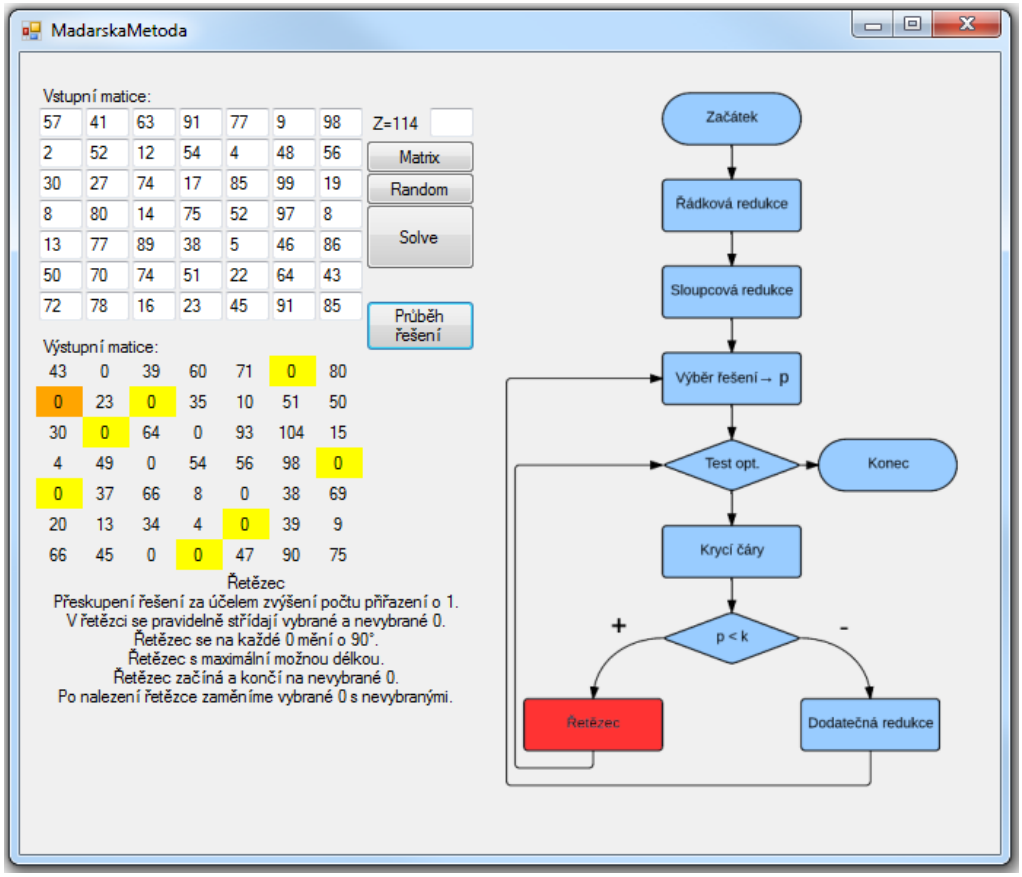
Obr. 5.9 – Výstup programu – Test optimality



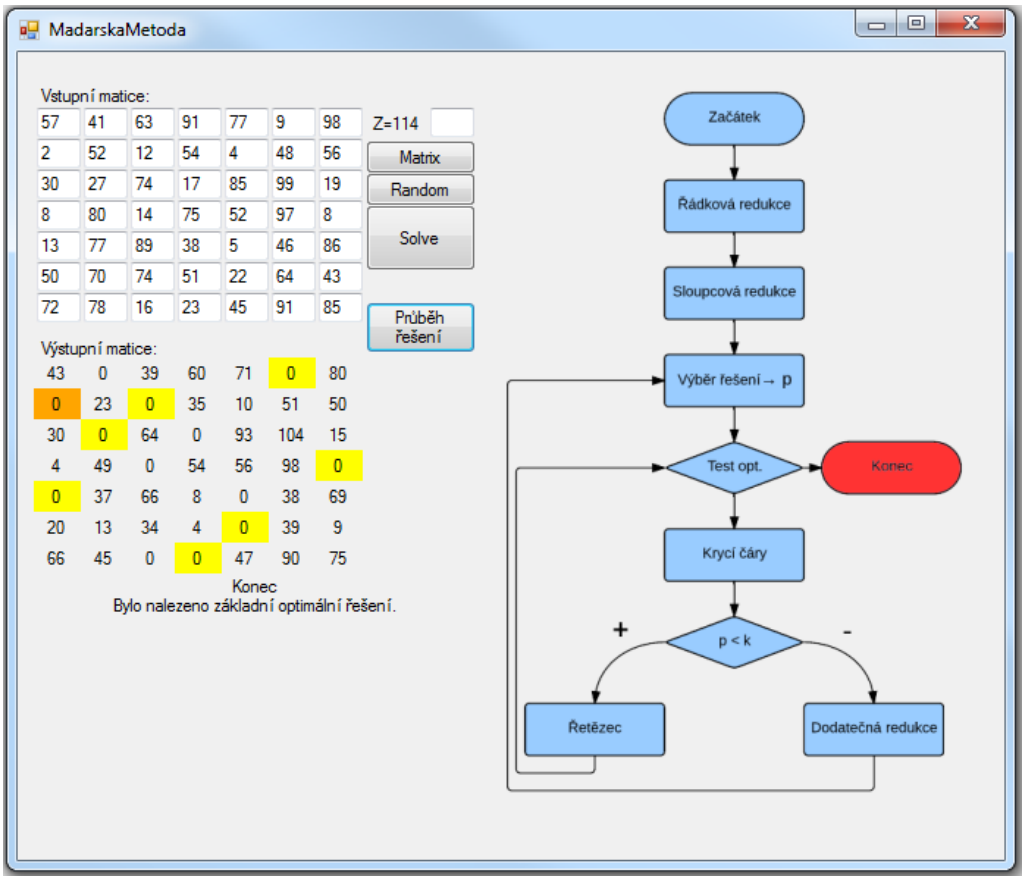
Obr. 5.10 – Výstup programu – Krycí čáry



Obr. 5.11 – Výstup programu – Dodatečná redukce



Obr. 5.12 – Výstup programu – Řetězec

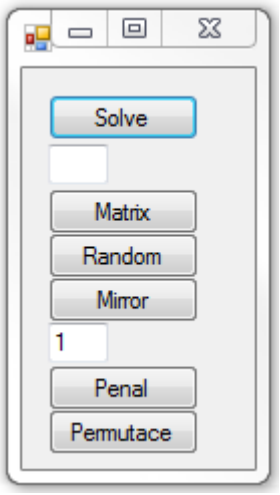


Obr. 5.13 – Výstup programu – Konec

5.5 Uživatelský manuál – Okružní dopravní problém

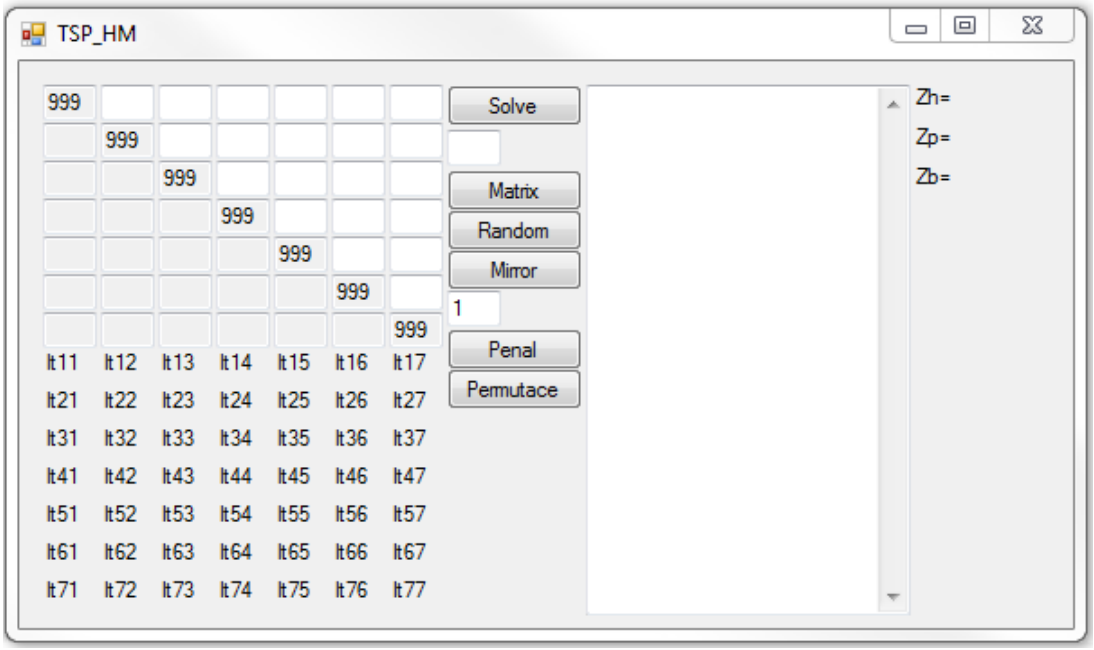
Ke spuštění je nutná instalace knihoven Microsoft Visual Studia, ve kterém byl program napsán. Ze složky programu nutno nainstalovat vcredist_x86.exe, obsahující výše zmíněné knihovny. Po instalaci, bude možno spustit výsledný program na jakékoli stanici s operačním systémem Windows x86/x64.

Jelikož se nejedná o výukový program, ale spíše o testovací prostředí pro odvozenou maďarskou metodu, grafické uživatelské rozhraní je zde zjednodušeno.



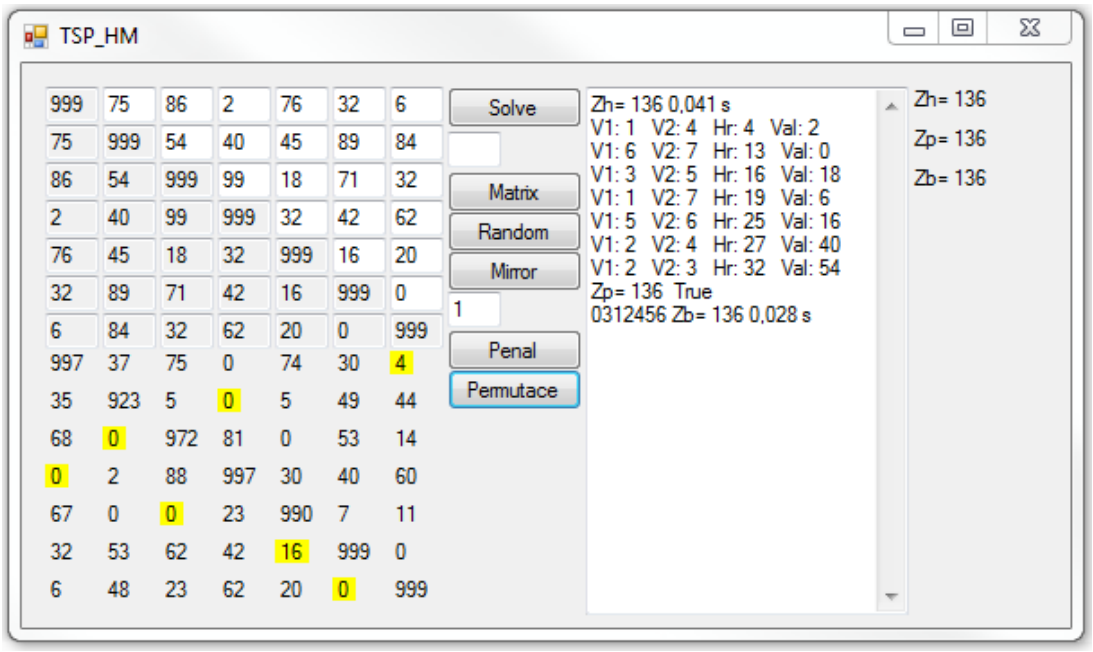
Obr. 5.14 – Úvodní spuštění programu

Stejně jako u předešlé verze po zadání velikosti, nastává vygenerování pracovní plochy programu *Obr. 5.15*. Změny jsou v zadávající matici, zadávána je pouze horní polovina, druhá část je automaticky doplněna pomocí tlačítka Mirror, tímto je zaručena zrcadlově souměrná matice, která je nutná k řešení okružního dopravního problému.



Obr. 5.15 - Vygenerovaná pracovní plocha s maticemi 7x7

Stejně jako v předešlém případě po vyplnění matice, dojde k vyřešení úlohy pomocí tlačítka Solve. Tímto dojde k vyřešení odvozenou maďarskou metodou, dále tlačítkem Penal, dojde k vyřešení metodou penalizací. Na závěr tlačítkem permutace, lze provést výpočet permutací všech možností, POZOR tuto metodu používat pouze do matic o velikosti 8x8, pro větší zadání exponenciální nárůst možností znemožní programu výpočet v rozumném čase.



Obr. 5.16 – Výstup programu – Zh – řešení odv. maď. metodou, Zp – řešení metodou penalizací, Zb – řešení metodou permutací

6 Výkonnost programu pro řešení ODP

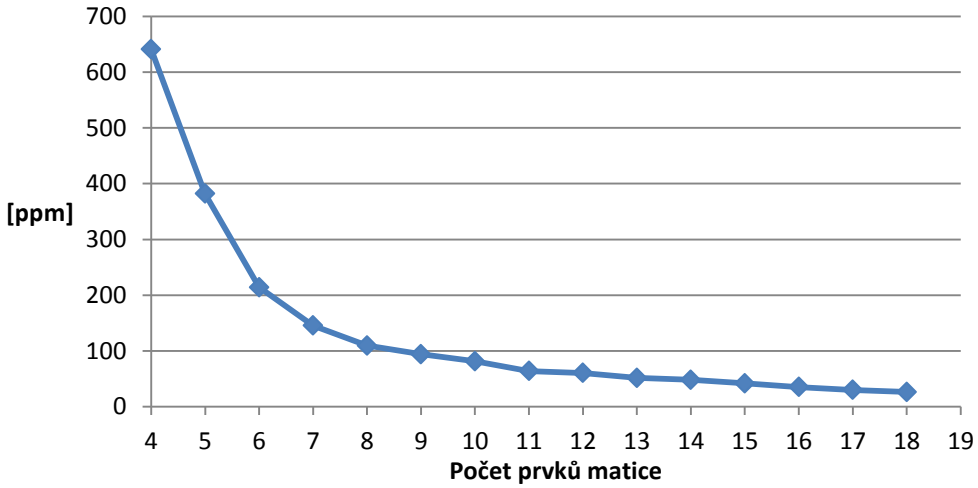
V této části zhodnotím celkový výkon praktické části, z hlediska časové náročnosti výpočtu, maximální výpočetní kapacitě, u metody penalizací procentuální úspěšností výpočtu a u odvozené maďarské metody maximální naměřenou chybu.

K měření bylo nutné použít exaktní metodu, která udává přesné výsledky. Nevýhodou těchto metod, jak již bylo výše zmíněno, je exponenciální náročnost výpočtu. V praktické části k tomuto byla použita metoda pro zjištění všech permutací, avšak složitostí kódu bylo reálné použít tuto metodu pouze do rozměru 8x8. Pro matice o velikosti 9x9 trval výpočet jedné úlohy několik desítek sekund. Pro určení chyby bylo vždy porovnáno jedno sto výsledků úloh. Z tohoto důvodu byla metoda permutací dosti nevyhovující, proto jako alternativa byla použita metoda penalizací.

Nevýhodou této metody je, že ne vždy je schopna určit výsledek dané úlohy, avšak pokud jej určí, tak vždy optimální. Její výpočetní rychlost nepřevyšuje jednotky sekund i při vyšších rozměrech matice. Proto byly použity pouze vygenerované matice, pro které lze určit výsledek právě metodou penalizací. Tím bylo zaručeno porovnávání s optimálním výsledkem. Avšak jak bylo vzápětí zjištěno, metoda penalizací není příliš univerzální, o čemž vypovídá její procentuální úspěšnost výpočtu.

Tab. 6.1 - Úspěšnost nalezení řešení metody penalizací

n = 1000	**4x4**	**5x5**	**6x6**	**7x7**	**8x8**	**9x9**	**10x10**	**11x11**
Gen. matic	15601	26184	46697	68790	91220	106378	122791	156735
Úspěšnost [ppm]	640,98	381,91	214,15	145,37	109,63	94,00	81,44	63,80
n = 1000	**12x12**	**13x13**	**14x14**	**15x15**	**16x16**	**17x17**	**18x18**	**19x19**
Gen. matic	165527	193793	207782	284650	284650	332645	382899	-
Úspěšnost [ppm]	60,41	51,60	48,13	35,13	35,13	30,06	26,12	-



Obr 6.1 - Úspěšnost metody penalizací

Pro porovnání s odvozenou maďarskou metodou, je k získání sta matic 10x10, které jsou řešitelné metodou penalizací třeba vygenerovat přibližně 12279 matic. Při testu odvozené maďarské metody, byly naměřeny tyto hodnoty.

Tab. 6.2 - Výkon odvezené maďarské metody

n = 100	7x7	8x8	9x9	10x10
		Čas výpočtu [s]		
Medián	0,057	0,1795	1,3985	4,8165
Průměr	0,06773	0,39582	4,84157	30,51373
Max	0,221	6,406	159,136	617,048
Min	0,031	0,064	0,153	0,108
		Chyba [%]		
Medián	1,028773	1,066746	1,093762	1,148359
Průměr	1,084275	1,120428	1,137002	1,20644
Max	1,643564	1,636364	1,596899	2,096491
Min	1	1	1	1

Na první pohled se může zdát, že odvozená maďarská metoda je vcelku rychlá, u matic 10x10 medián výpočetního času vyšel 4,8 sekundy, horší už je průměrná hodnota výpočetního času 30,5 sekundy. Toto navýšení vzniklo kvůli třem extrémním hodnotám z měření odpovídající 295, 610 a 617 sekund. Tyto velice odlišné hodnoty vznikly z důvodu, kdy při výběru řešení, bylo třeba vybrat více než tři prvky větší než nula.

Určení chyby je vyjádřeno násobkem optimálního řešení, v tomto směru disponuje tato metoda vcelku dobrými výsledky. Maximální chyba byla naměřena 2,09, průměrná hodnota 1,2 a medián pouze 1,14. Úspěšnost určení optimálního řešení byla 21% u matic 10x10 a 34% u matic 9x9.

Závěr

Úvodní část práce obsahuje obecný popis problematiky optimalizačních úloh. Zavádí pojmy účelová funkce, minimalizace resp. maximalizace, definující a omezující podmínky, optimální základní řešení a přípustné řešení. Dále se zabývá distribučními úlohami, jejich typy a metodami jejich řešení.

Mezi ně patří přiřazovací problém, jde o speciální případ jednostupňové dopravní úlohy, řešený tzv. Maďarskou metodou. Řešení Maďarské metody se realizuje zejména pomocí těchto kroků: Řádková a sloupcová redukce, určení incidenčních indexů, zavedení krycích čar, výběr řešení, dodatečná redukce a řetězec. Přesný postup je popsán a názorně předveden na příkladu v kapitole *3.1.2*.

Praktická část přiřazovacího problému maďarskou metodou je zpracována jako výukový program s intuitivním uživatelským rozhraním a kompletním rozborem problematiky. Teoretická část obsahuje mimo jiné uživatelský manuál pro snadné ovládání programu.

Další část práce je věnována problematice okružní dopravní úlohy („problém obchodního cestujícího“). V teoretické části je diskutována náročnost těchto úloh (jako NP-úplná), jelikož pro tento typ úloh neexistuje algoritmus, který by pracoval v polynomiálním čase a byl schopen vždy poskytnout základní optimální řešení. Dle nalezené literatury není možné sestavit exaktní algoritmus, splňující tyto podmínky.

Pro výpočet okružního dopravního problému byly zpracovány následující metody: metoda penalizací, metoda permutací a odvozená Maďarská metoda. Metoda penalizací je metoda definována v oblasti teorie grafů, výhodou této metody je výpočet v polynomiálním čase, nevýhodou nejistota získání výsledku. Mohou nastat dva stavy, metoda buď dá optimální výsledek nebo vůbec nenalezne řešení. Při testech byla zjištěna relativně malá úspěšnost této metody, při velkých rozměrech úlohy dosahovala pouze desítek ppm. Proto tato metoda byla použita pouze ke generování zadání, na které znala řešení, a tato řešení byla porovnávána s výsledky dalších metod.

Metoda permutací pracuje na principu nalezení všech okružních cest a výběru nejkratší možné. Tato metoda je typickým zástupce algoritmu s exponenciální závislostí výpočetní náročnosti, a proto její použití bylo možné pouze do matic o velikosti cca 8x8. Obě předešlé metody poskytovaly optimální řešení, avšak nebyly univerzální.

Kompromisně byla zvolena metoda, odvozená od Maďarské metody. Podstatou této metody je nalezení optimálního řešení pro přiřazovací problém. To však v praxi u

okružního dopravního problému přináší možnost tzv. parciálních smyček. Metoda postupně zařazuje do řešení nejmenší nenulové prvky, což může v nešťastných případech zapříčinit nalezení horšího než optimálního přípustného řešení. Výsledky této metody jsou nicméně uspokojivé, průměrná chyba této metody je velice nízká i při vyšších rozměrech matice a totéž platí i o rychlosti a efektivnosti metody.

Seznam literatury a informačních zdrojů

[1] KUČERA, Luděk. 1989. *Kombinatorické algoritmy*. 2., nezm. vyd. Praha: Státní nakladatelství technické literatury, 286 s. Matematický seminář SNTL.

[2] *Maďarska metoda* [online]. [cit. 2014-12-17]. Dostupné z: pef.czu.cz/~houska/EMM_KS/Materialy/Prednasky/Konzultace_II.ppt

[3] KOSKOVÁ, Ing. Ivanka. 2007. *Distribuční úlohy I*. Praha: Česká zemědělská univerzita v Praze.

[4] *Matematické modely dopravní úlohy* [online]. [cit. 2015-05-10]. Dostupné z: homel.vsb.cz/~dor028/Modely_dopravni_ulohy.doc

[5] *Přiřazovací problém - turnovfree.net* [online]. [cit. 2014-12-17]. Dostupné z: www.turnovfree.net/~stybla/skola/czu/tretak/.../prirazovaci_problem.pdf

[6] PELIKÁN, Jan, Jan FÁBRY, Tomáš VYMYSLICKÝ, Jan PELIKÁN, Pavlína GOTTWALDOVÁ a Jan NEDĚLNÍK. *Heuristics for routes generation in pickup and delivery problem*. ISBN 10.1007/978-90-481-8706-5_24.

[7] JABLONSKÝ, Josef. 2002. *Operační výzkum: kvantitativní modely pro ekonomické rozhodování*. 2. vyd. Praha: Professional Publishing, 323 s. ISBN Programy pro matematické modelování.

[8] JABLONSKÝ, Josef. 2007. *Programy pro matematické modelování*. Vyd. 1. Praha: Oeconomica, 258 s. ISBN 978-80-245-1178-8.

[9] *Základy operačního výzkumu - Katedra ekonometrie* [online]. [cit. 2014-12-17]. Dostupné z: k101.unob.cz/~neubauer/pdf/zov11.pdf

[10] ČERNÝ, Jakub. 2010. *Základní grafové algoritmy.- Katedra Aplikované Matematiky* [online]. [cit. 2015-05-10]. Dostupné z: http://kam.mff.cuni.cz/~kuba/ka/ka.pdf

[11] *Maďarská metoda - Operační výzkum - Dopravní fakulta ...* [online]. [cit. 2014-12-17]. Dostupné z: www.primat.cz/upce-dfjp/predmety/operacni.../madarska-metoda/133547

[12] PELIKÁN, Jan. 2001. *Diskrétní modely v operačním výzkumu*. 1.vyd. Praha: Professional Publishing, 163 s. ISBN c++.

[13] PROKOP, Jiří. 2012. *Algoritmy v jazyku C a C*. 2., rozš. a aktualiz. vyd. Praha: Grada, 169 s. Průvodce (Grada). ISBN c++.

[14] ŠÍMA, František a David VILÍMEK. 2006. *Microsoft Visual Studio .NET: praktické programování krok za krokem*. 1. vyd. Praha: Grada, 254 s. Průvodce (Grada). ISBN algoritmy.

[15] FORD, Sara a Mark SUMMERFIELD. 2009. *266 tipů a triků pro Microsoft Visual Studio: Prentice Hall Open Source Software Development Series*. Vyd. 1. Brno:

Computer Press, 224 s. ISBN 978-80-251-2554-0.

[16] *Teorie grafù a diskrétní optimalizace 2* [online]. 2009. RYJÁČEK, Z. [cit. 2015-05-10]. Dostupné z: http://www.kma.zcu.cz/TGD2

[17] KUMAR, Dr. D. Nagesh. *Linear Programming Applications - Assignment Problem* [online]. [cit. 2015-05-10]. Dostupné z: http://nptel.ac.in/courses/105108127/pdf/Module_4/M4L3slides.pdf

[18] *Travelling salesman problem* [online]. CHHABRA, Vinay. [cit. 2015-05-10]. Dostupné z: http://www.universalteacherpublications.com/univ/ebooks/or/Ch6/travsales.htm

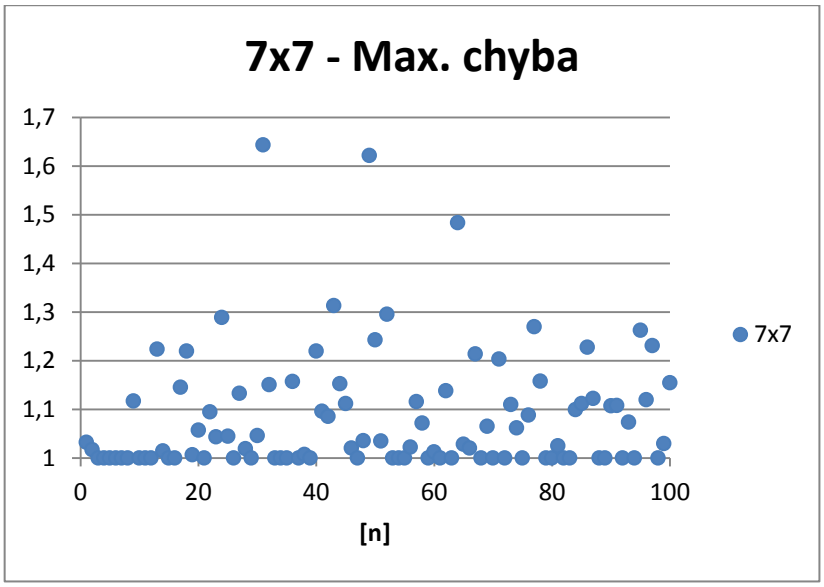
Přílohy

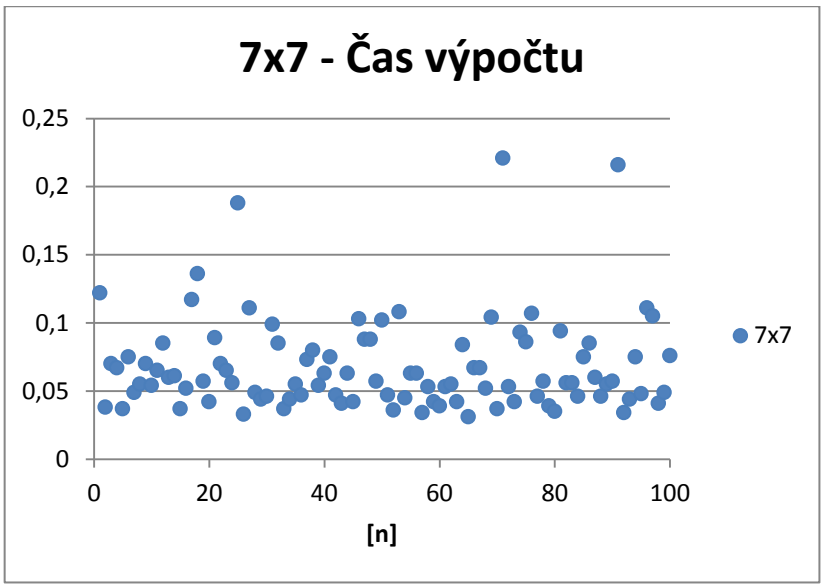
Matice 7x7 – 100 testovacích, náhodně vygenerovaných matic

Zh	čas [s]	Zpenal	čas [s]	Zperm	dif	chyba
160	0,122	155	0,03	155	5	**1,0322581**
175	0,038	172	0,03	172	3	**1,0174419**
132	0,07	132	0,029	132	0	**1**
199	0,067	199	0,028	199	0	**1**
256	0,037	256	0,028	256	0	**1**
250	0,075	250	0,028	250	0	**1**
153	0,049	153	0,028	153	0	**1**
199	0,055	199	0,029	199	0	**1**
191	0,07	171	0,028		20	**1,1169591**
156	0,054	156	0,031	156	0	**1**
88	0,065	88	0,031	88	0	**1**
194	0,085	194	0,03	194	0	**1**
164	0,06	134	0,028	134	30	**1,2238806**
214	0,061	211	0,029	211	3	**1,014218**
171	0,037	171	0,029		0	**1**
111	0,052	111	0,029	111	0	**1**
284	0,117	248	0,029	248	36	**1,1451613**
233	0,136	191	0,029	191	42	**1,2198953**
159	0,057	158	0,028	158	1	**1,0063291**
148	0,042	140	0,03		8	**1,0571429**
189	0,089	189	0,029		0	**1**
197	0,07	180	0,028	180	17	**1,0944444**
168	0,065	161	0,029	161	7	**1,0434783**
165	0,056	128	0,03	128	37	**1,2890625**
141	0,188	135	0,029	135	6	**1,0444444**
124	0,033	124	0,029	124	0	**1**
239	0,111	211	0,03	211	28	**1,1327014**
159	0,049	156	0,03	156	3	**1,0192308**
147	0,044	147	0,029	147	0	**1**
251	0,046	240	0,03		11	**1,0458333**
166	0,099	101	0,03	101	65	**1,6435644**
275	0,085	239	0,03	239	36	**1,1506276**
76	0,037	76	0,029	76	0	**1**
130	0,044	130	0,029	130	0	**1**
223	0,055	223	0,03	223	0	**1**
316	0,047	273	0,03		43	**1,1575092**
245	0,073	245	0,03		0	**1**
267	0,08	265	0,029	265	2	**1,0075472**
299	0,054	299	0,031		0	**1**
233	0,063	191	0,031	191	42	**1,2198953**
308	0,075	281	0,03	281	27	**1,0960854**
114	0,047	105	0,031	105	9	**1,0857143**
130	0,041	99	0,03	99	31	**1,3131313**
166	0,063	144	0,031	144	22	**1,1527778**
209	0,042	188	0,03	188	21	**1,1117021**

252	0,103	247	0,029	247	5	**1,0202429**
190	0,088	190	0,03	190	0	**1**
174	0,088	168	0,03	168	6	**1,0357143**
180	0,057	111	0,03	111	69	**1,6216216**
225	0,102	181	0,029	181	44	**1,2430939**
207	0,047	200	0,03	200	7	**1,035**
228	0,036	176	0,028	176	52	**1,2954545**
212	0,108	212	0,028	212	0	**1**
118	0,045	118	0,029	118	0	**1**
171	0,063	171	0,028		0	**1**
271	0,063	265	0,028	265	6	**1,0226415**
125	0,034	112	0,028	112	13	**1,1160714**
120	0,053	112	0,027		8	**1,0714286**
201	0,042	201	0,029	201	0	**1**
163	0,039	161	0,028	161	2	**1,0124224**
235	0,053	235	0,029	235	0	**1**
181	0,055	159	0,028		22	**1,1383648**
203	0,042	203	0,029		0	**1**
184	0,084	124	0,03	124	60	**1,483871**
110	0,031	107	0,028	107	3	**1,0280374**
198	0,067	194	0,028		4	**1,0206186**
227	0,067	187	0,029		40	**1,2139037**
146	0,052	146	0,03	146	0	**1**
278	0,104	261	0,028	261	17	**1,0651341**
128	0,037	128	0,029	128	0	**1**
349	0,221	290	0,029	290	59	**1,2034483**
143	0,053	143	0,029		0	**1**
293	0,042	264	0,031	264	29	**1,1098485**
224	0,093	211	0,029	211	13	**1,0616114**
233	0,086	233	0,031	233	0	**1**
210	0,107	193	0,028	193	17	**1,0880829**
226	0,046	178	0,029	178	48	**1,2696629**
264	0,057	228	0,029		36	**1,1578947**
204	0,039	204	0,029	204	0	**1**
148	0,035	148	0,029	148	0	**1**
248	0,094	242	0,029		6	**1,0247934**
162	0,056	162	0,029	162	0	**1**
155	0,056	155	0,028	155	0	**1**
199	0,046	181	0,029	181	18	**1,0994475**
169	0,075	152	0,03	152	17	**1,1118421**
205	0,085	167	0,029	167	38	**1,2275449**
174	0,06	155	0,029		19	**1,1225806**
242	0,046	242	0,03	242	0	**1**
156	0,055	156	0,03	156	0	**1**
258	0,057	233	0,03	233	25	**1,1072961**
113	0,216	102	0,03		11	**1,1078431**
266	0,034	266	0,029	266	0	**1**
102	0,044	95	0,03	95	7	**1,0736842**
79	0,075	79	0,03	79	0	**1**
149	0,048	118	0,03		31	**1,2627119**
206	0,111	184	0,029	184	22	**1,1195652**

240	0,105	195	0,03		45	**1,2307692**
173	0,041	173	0,029	173	0	**1**
314	0,049	305	0,03	305	9	**1,0295082**
209	0,076	181	0,031	181	28	**1,1546961**

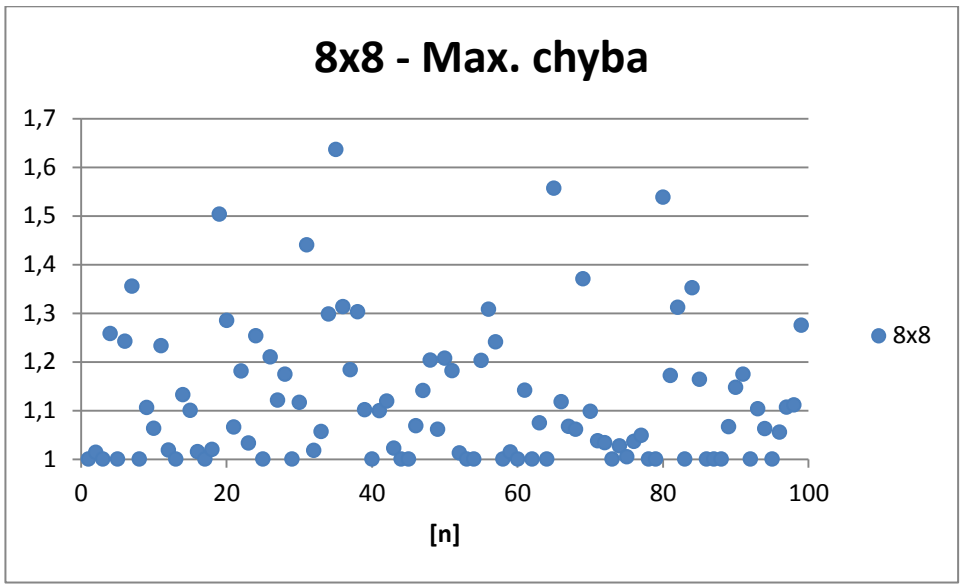


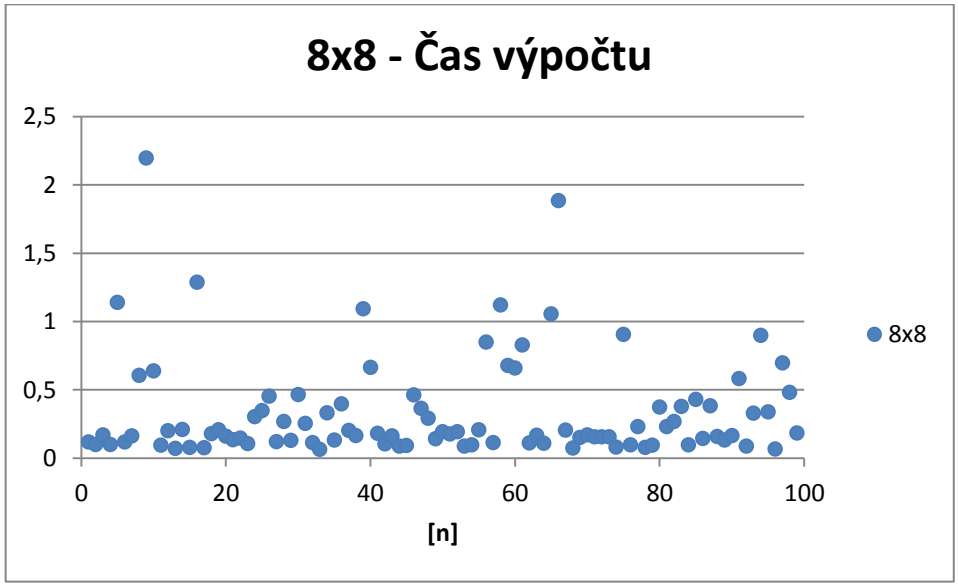


Matice 8x8 – 100 testovacích, náhodně vygenerovaných matic

Zh	čas [s]	Zpenal	čas [s]	Zperm	dif	chyba
142	0,118	142	0,537	142	0	**1**
149	0,1	147	0,544	147	2	**1,013605**
218	0,17	218	0,529		0	**1**
200	0,099	159	0,552		41	**1,257862**
184	1,14	184	0,562	184	0	**1**
200	0,119	161	0,554		39	**1,242236**
206	0,162	152	0,539	152	54	**1,355263**
354	0,606	354	0,546	354	0	**1**
250	2,195	226	0,533		24	**1,106195**
135	0,638	127	0,558	127	8	**1,062992**
217	0,094	176	0,496	176	41	**1,232955**
337	0,2	331	0,544	331	6	**1,018127**
187	0,07	187	0,547	187	0	**1**
223	0,21	197	0,55	197	26	**1,13198**
154	0,077	140	0,519		14	**1,1**
205	1,287	202	0,563		3	**1,014851**
256	0,076	256	0,545	256	0	**1**
261	0,178	256	0,562	256	5	**1,019531**
239	0,207	159	0,542	159	80	**1,503145**
203	0,161	158	0,53	158	45	**1,28481**
242	0,134	227	0,525	227	15	**1,066079**
196	0,145	166	0,547	166	30	**1,180723**
158	0,107	153	0,51		5	**1,03268**
203	0,303	162	0,567		41	**1,253086**
247	0,347	247	0,557	247	0	**1**
225	0,453	186	0,528	186	39	**1,209677**
194	0,12	173	0,525	173	21	**1,121387**
249	0,268	212	0,505	212	37	**1,174528**
151	0,13	151	0,571	151	0	**1**
221	0,464	198	0,533	198	23	**1,116162**
193	0,254	134	0,531		59	**1,440299**
174	0,114	171	0,527		3	**1,017544**
206	0,064	195	0,539		11	**1,05641**
322	0,331	248	0,539		74	**1,298387**
144	0,131	88	0,577	88	56	**1,636364**
172	0,396	131	0,573	131	41	**1,312977**
174	0,203	147	0,527		27	**1,183673**
267	0,164	205	0,568		62	**1,302439**
174	1,092	158	0,556	158	16	**1,101266**
152	0,664	152	0,566	152	0	**1**
155	0,181	141	0,528	141	14	**1,099291**
226	0,105	202	0,51		24	**1,118812**
229	0,162	224	0,558		5	**1,022321**
261	0,087	261	0,516	261	0	**1**
208	0,092	208	0,536		0	**1**
171	0,462	160	0,545		11	**1,06875**
267	0,363	234	0,56	234	33	**1,141026**
308	0,291	256	0,52		52	**1,203125**
243	0,142	229	0,541	229	14	**1,061135**

262	0,193	217	0,539	217	45	**1,207373**
286	0,177	242	0,528		44	**1,181818**
242	0,192	239	0,564		3	**1,012552**
189	0,087	189	0,532	189	0	**1**
219	0,098	219	0,528	219	0	**1**
273	0,206	227	0,493	227	46	**1,202643**
204	0,849	156	0,526		48	**1,307692**
232	0,113	187	0,53	187	45	**1,240642**
107	1,121	107	0,509	107	0	**1**
141	0,679	139	0,578	139	2	**1,014388**
135	0,659	135	0,5		0	**1**
291	0,827	255	0,518	255	36	**1,141176**
194	0,111	194	0,544	194	0	**1**
202	0,168	188	0,524	188	14	**1,074468**
155	0,108	155	0,552	155	0	**1**
165	1,056	106	0,547		59	**1,556604**
171	1,885	153	0,582		18	**1,117647**
127	0,204	119	0,514		8	**1,067227**
104	0,074	98	0,552	98	6	**1,061224**
222	0,15	162	0,557	162	60	**1,37037**
168	0,169	153	0,552		15	**1,098039**
221	0,156	213	0,567	213	8	**1,037559**
186	0,155	180	0,512		6	**1,033333**
217	0,156	217	0,53	217	0	**1**
229	0,081	223	0,559	223	6	**1,026906**
226	0,906	225	0,522		1	**1,004444**
202	0,097	195	0,554	195	7	**1,035897**
172	0,231	164	0,538	164	8	**1,04878**
210	0,077	210	0,523	210	0	**1**
100	0,095	100	0,528	100	0	**1**
160	0,373	104	0,517		56	**1,538462**
184	0,231	157	0,553		27	**1,171975**
223	0,267	170	0,572	170	53	**1,311765**
161	0,377	161	0,555	161	0	**1**
146	0,098	108	0,559		38	**1,351852**
234	0,43	201	0,551	201	33	**1,164179**
299	0,144	299	0,544	299	0	**1**
200	0,383	200	0,541	200	0	**1**
149	0,158	149	0,566	149	0	**1**
177	0,132	166	0,552	166	11	**1,066265**
202	0,164	176	0,536	176	26	**1,147727**
202	0,582	172	0,536	172	30	**1,174419**
195	0,087	195	0,577		0	**1**
288	0,329	261	0,544		27	**1,103448**
256	0,899	241	0,584		15	**1,062241**
160	0,339	160	0,565	160	0	**1**
325	0,066	308	0,569	308	17	**1,055195**
187	0,697	169	0,512		18	**1,106509**
240	0,481	216	0,559	216	24	**1,111111**
190	0,183	149	0,535	149	41	**1,275168**
174	6,406	174	0,518	174	0	**1**

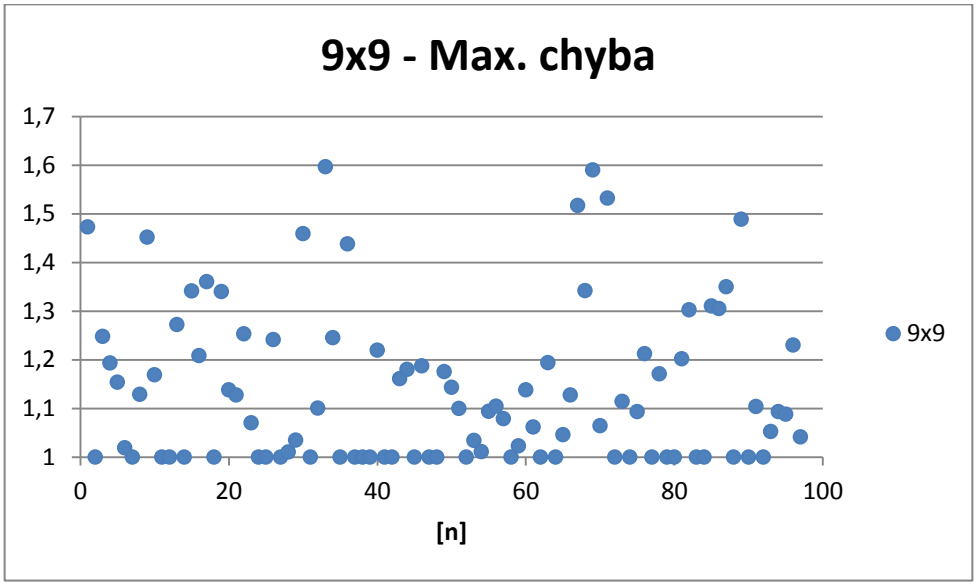

8x8 - Max. chyba
1,7
1,6
1,5
1,4
1,3
1,2
1,1
1
0
20
40
60
80
100
[n]
8x8

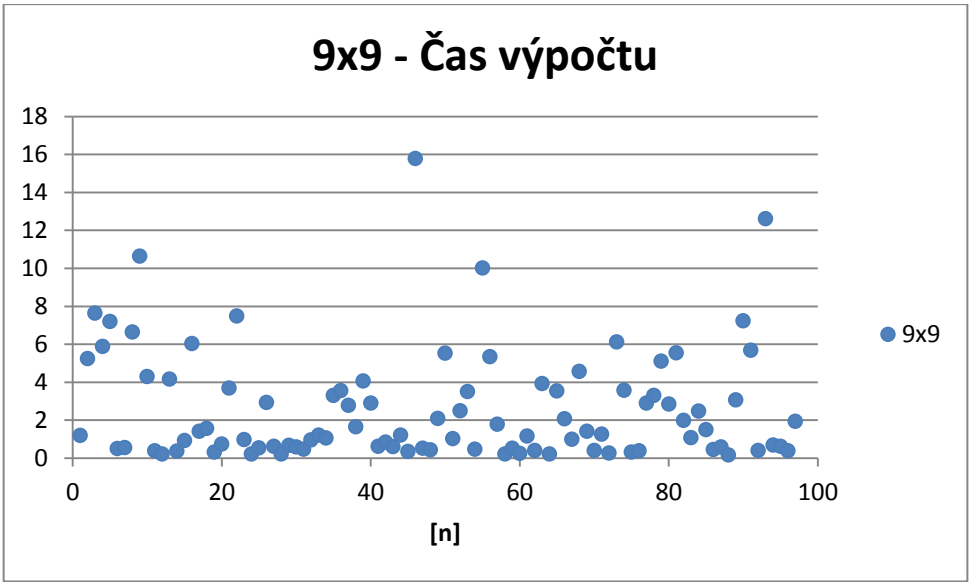

8x8 - Čas výpočtu
2,5
2
1,5
1
0,5
0
0
20
40
60
80
100
[n]
8x8

Matice 9x9 – 100 testovacích, náhodně vygenerovaných matic

Zh	čas [s]	Zpenal	čas [s]	dif	chyba
277	1,194	188	0,002	89	**1,473404**
237	5,242	237	0,001	0	**1**
282	7,627	226	0,002	56	**1,247788**
216	5,868	181	0,002	35	**1,19337**
210	7,187	182	0,003	28	**1,153846**
211	0,5	207	0,002	4	**1,019324**
207	0,54	207	0,003	0	**1**
332	6,638	294	0,002	38	**1,129252**
196	10,628	135	0,002	61	**1,451852**
242	4,29	207	0,002	35	**1,169082**
191	0,379	191	0,002	0	**1**
124	0,216	124	0,002	0	**1**
229	4,156	180	0,003	49	**1,272222**
244	0,362	244	0,003	0	**1**
165	0,916	123	0,003	42	**1,341463**
220	6,022	182	0,004	38	**1,208791**
351	1,399	258	0,003	93	**1,360465**
205	1,565	205	0,003	0	**1**
130	0,304	97	0,004	33	**1,340206**
206	0,738	181	0,004	25	**1,138122**
168	3,675	149	0,005	19	**1,127517**
272	7,486	217	0,003	55	**1,253456**
335	0,961	313	0,002	22	**1,070288**
175	0,211	175	0,003	0	**1**
168	0,526	168	0,003	0	**1**
216	2,917	174	0,005	42	**1,241379**
170	0,609	170	0,003	0	**1**
284	0,204	281	0,006	3	**1,010676**
265	0,671	256	0,005	9	**1,035156**
197	0,576	135	0,005	62	**1,459259**
182	0,468	182	0,008	0	**1**
208	0,951	189	0,005	19	**1,100529**
206	1,208	129	0,005	77	**1,596899**
259	1,053	208	0,004	51	**1,245192**
158	3,293	158	0,006	0	**1**
197	3,546	137	0,006	60	**1,437956**
159	2,77	159	0,007	0	**1**
261	1,65	261	0,006	0	**1**
144	4,052	144	0,006	0	**1**
300	2,892	246	0,005	54	**1,219512**
191	0,621	191	0,006	0	**1**
155	0,832	155	0,009	0	**1**
187	0,597	161	0,008	26	**1,161491**
229	1,198	194	0,006	35	**1,180412**
163	0,336	163	0,009	0	**1**
266	15,779	224	0,006	42	**1,1875**
130	0,517	130	0,008	0	**1**
264	0,435	264	0,006	0	**1**
194	2,081	165	0,008	29	**1,175758**

199	5,521	174	0,009	25	**1,143678**
165	1,016	150	0,007	15	**1,1**
287	2,478	287	0,009	0	**1**
212	3,493	205	0,008	7	**1,034146**
278	0,454	275	0,007	3	**1,010909**
279	10,005	255	0,008	24	**1,094118**
158	5,331	143	0,011	15	**1,104895**
191	1,78	177	0,008	14	**1,079096**
139	0,204	139	0,007	0	**1**
180	0,511	176	0,007	4	**1,022727**
264	0,246	232	0,01	32	**1,137931**
274	1,157	258	0,009	16	**1,062016**
158	0,386	158	0,009	0	**1**
234	3,92	196	0,009	38	**1,193878**
140	0,201	140	0,01	0	**1**
136	3,537	130	0,009	6	**1,046154**
159	2,065	141	0,008	18	**1,12766**
223	0,985	147	0,007	76	**1,517007**
196	4,558	146	0,011	50	**1,342466**
264	1,398	166	0,012	98	**1,590361**
181	0,39	170	0,008	11	**1,064706**
236	1,26	154	0,008	82	**1,532468**
149	0,26	149	0,01	0	**1**
234	6,115	210	0,013	24	**1,114286**
206	3,565	206	0,01	0	**1**
199	0,315	182	0,008	17	**1,093407**
171	0,378	141	0,008	30	**1,212766**
170	2,885	170	0,012	0	**1**
322	3,3	275	0,009	47	**1,170909**
142	5,093	142	0,01	0	**1**
171	2,834	171	0,01	0	**1**
357	5,535	297	0,009	60	**1,20202**
211	1,976	162	0,01	49	**1,302469**
126	1,064	126	0,007	0	**1**
204	2,469	204	0,007	0	**1**
291	1,493	222	0,006	69	**1,310811**
141	0,446	108	0,006	33	**1,305556**
189	0,586	140	0,005	49	**1,35**
233	0,153	233	0,006	0	**1**
326	3,06	219	0,014	107	**1,488584**
188	7,232	188	0,013	0	**1**
298	5,665	270	0,014	28	**1,103704**
159	0,39	159	0,011	0	**1**
179	12,609	170	0,012	9	**1,052941**
246	0,676	225	0,012	21	**1,093333**
234	0,607	215	0,017	19	**1,088372**
203	0,375	165	0,018	38	**1,230303**
126	1,937	121	0,014	5	**1,041322**
205	159,136	205	0,004	0	**1**
177	39,998	158	0,009	19	**1,120253**
228	35,223	188	0,003	40	**1,212766**

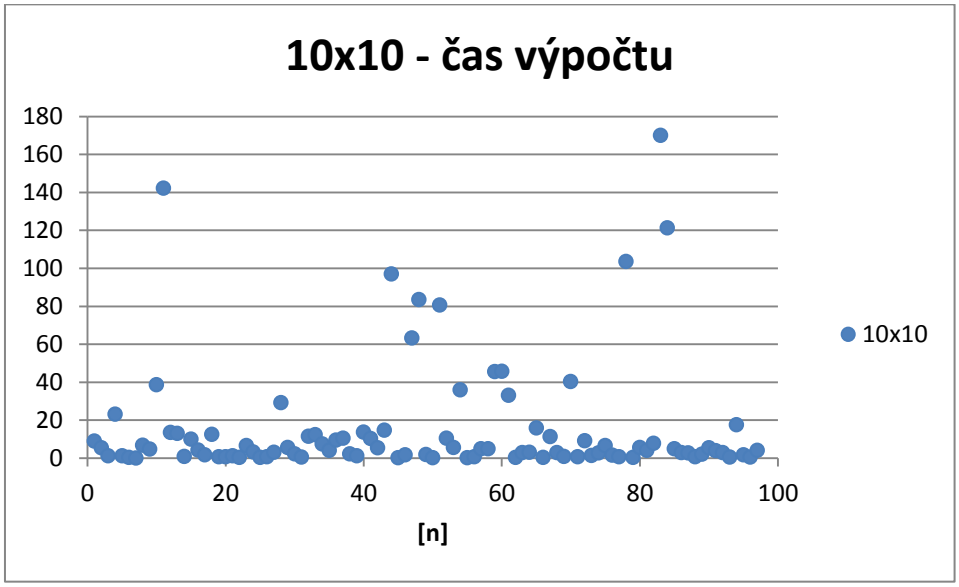

9x9 - Max. chyba
1,7
1,6
1,5
1,4
1,3
1,2
1,1
1
0
20
40
60
80
100
[n]
9x9

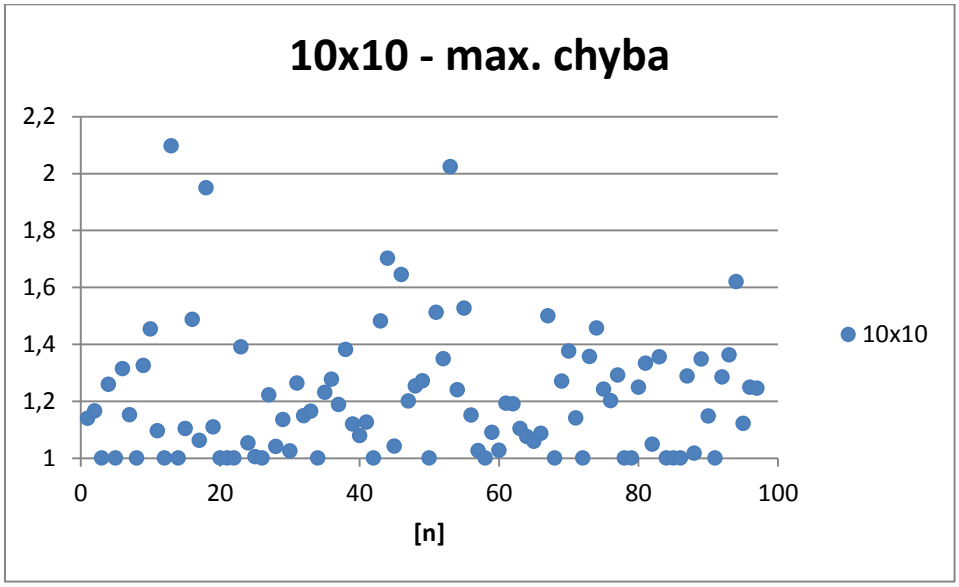

9x9 - Čas výpočtu
18
16
14
12
10
8
6
4
2
0
0
20
40
60
80
100
[n]
9x9

Matice 10x10 – 100 testovacích, náhodně vygenerovaných matic

Zh	čas [s]	Zpenal	čas [s]	dif	chyba
222	0,004	253	8,993	31	**1,13964**
254	0,001	296	5,499	42	**1,165354**
141	0,002	141	1,221	0	**1**
162	0,002	204	23,116	42	**1,259259**
179	0,001	179	1,238	0	**1**
153	0,001	201	0,473	48	**1,313725**
164	0,001	189	0,108	25	**1,152439**
225	0,002	225	6,8	0	**1**
172	0,001	228	4,748	56	**1,325581**
163	0,001	237	38,752	74	**1,453988**
198	0,004	217	142,245	19	**1,09596**
108	0,002	108	13,551	0	**1**
114	0,002	239	13,099	125	**2,096491**
154	0,002	154	0,952	0	**1**
271	0,002	299	10,038	28	**1,103321**
156	0,003	232	4,229	76	**1,487179**
161	0,002	171	1,757	10	**1,062112**
119	0,002	232	12,465	113	**1,94958**
128	0,002	142	0,678	14	**1,109375**
156	0,004	156	0,753	0	**1**
164	0,001	164	1,22	0	**1**
90	0,002	90	0,327	0	**1**
164	0,003	228	6,66	64	**1,390244**
151	0,002	159	3,201	8	**1,05298**
199	0	200	0,45	1	**1,005025**
142	0,001	142	0,795	0	**1**
126	0,001	154	3,09	28	**1,222222**
121	0,001	126	29,266	5	**1,041322**
244	0,002	277	5,643	33	**1,135246**
196	0,002	201	2,333	5	**1,02551**
133	0,001	168	0,647	35	**1,263158**
289	0,001	332	11,462	43	**1,148789**
195	0,003	227	12,428	32	**1,164103**
174	0,001	174	7,427	0	**1**
217	0,002	267	4,037	50	**1,230415**
173	0,003	221	9,529	48	**1,277457**
218	0,001	259	10,47	41	**1,188073**
178	0,001	246	2,236	68	**1,382022**
167	0,002	187	1,249	20	**1,11976**
215	0,001	232	13,753	17	**1,07907**
150	0,002	169	10,42	19	**1,126667**
237	0,001	237	5,385	0	**1**
218	0,001	323	14,663	105	**1,481651**
84	0,002	143	97,001	59	**1,702381**
193	0,003	201	0,246	8	**1,041451**
93	0,002	153	1,813	60	**1,645161**
230	0,001	276	63,259	46	**1,2**
256	0,002	321	83,598	65	**1,253906**
107	0,002	136	1,98	29	**1,271028**

252	0,001	252	0,234	0	**1**
129	0	195	80,758	66	**1,511628**
172	0,001	232	10,441	60	**1,348837**
84	0,001	170	5,678	86	**2,02381**
192	0,002	238	35,908	46	**1,239583**
133	0,001	203	0,168	70	**1,526316**
231	0,001	266	0,799	35	**1,151515**
263	0,001	270	5,003	7	**1,026616**
97	0,001	97	4,951	0	**1**
221	0,001	241	45,619	20	**1,090498**
292	0	300	45,808	8	**1,027397**
229	0,001	273	33,156	44	**1,19214**
142	0,001	169	0,356	27	**1,190141**
290	0,001	320	2,976	30	**1,103448**
171	0,001	184	3,102	13	**1,076023**
221	0,001	234	15,849	13	**1,058824**
127	0,002	138	0,431	11	**1,086614**
136	0,001	204	11,408	68	**1,5**
210	0,001	210	2,976	0	**1**
189	0,001	240	0,854	51	**1,269841**
178	0,001	245	40,337	67	**1,376404**
170	0,001	194	0,73	24	**1,141176**
284	0,001	284	9,197	0	**1**
140	0,001	190	1,386	50	**1,357143**
94	0,002	137	2,762	43	**1,457447**
136	0,001	169	6,607	33	**1,242647**
179	0,001	215	1,599	36	**1,201117**
168	0,002	217	0,786	49	**1,291667**
194	0,002	194	103,701	0	**1**
205	0,001	205	0,454	0	**1**
221	0	276	5,606	55	**1,248869**
174	0,001	232	4,127	58	**1,333333**
183	0,001	192	7,868	9	**1,04918**
208	0,001	282	170,107	74	**1,355769**
189	0,001	189	121,406	0	**1**
167	0,001	167	4,885	0	**1**
175	0,001	175	2,904	0	**1**
149	0,002	192	2,803	43	**1,288591**
225	0,001	229	0,73	4	**1,017778**
158	0,001	213	2,099	55	**1,348101**
169	0,001	194	5,431	25	**1,147929**
102	0,001	102	3,859	0	**1**
130	0,001	167	2,885	37	**1,284615**
160	0,001	218	0,54	58	**1,3625**
163	0,001	264	17,654	101	**1,619632**
188	0,001	211	1,702	23	**1,12234**
161	0,002	201	0,487	40	**1,248447**
224	0,002	279	4,076	55	**1,245536**
199	0,002	225	610,464	26	**1,130653**
111	0,001	120	617,048	9	**1,081081**
196	0,001	217	295,355	21	**1,107143**


10x10 - čas výpočtu
180
160
140
120
100
80
60
40
20
0
0
20
40
60
80
100
[n]
10x10


10x10 - max. chyba
2,2
2
1,8
1,6
1,4
1,2
1
0
20
40
60
80
100
[n]
10x10