Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Umelá Inteligencia

Zadanie č.1

g) Eulerov kôň s Warnnsdorfovou heuristikou

Akademický rok 2022/2023

Meno: Ján Ágh
Cvičiaci: Ing. Martin Komák, PhD.

Dátum: 8.10.2022
Počet strán: 10

Obsah

1 Stručný opis problematiky a zadania 1
2 Navrhnuté riešenie 2
2.1 Použité programové prostriedky 2
2.2 Organizácia projektu 2
2.3 Diagram algoritmu prehľadávania 3
2.4 Slovný opis algoritmu prehľadávania 4
2.5 Možné výsledky prehľadávania 5
3 Testovanie správnosti 7
3.1 Spôsob testovania 7
3.1.1 Testovanie šachovnice 5x5 7
3.1.2 Testovanie šachovnice 8x8 8
3.1.3 Testovanie šachovnice väčších rozmerov 9
3.2 Zhodnotenie testovania 10
3.3 Možnosti rozšírenia a optimalizácie riešenia 10

1 Stručný opis problematiky a zadania

Cieľom úlohy je navrhnúť program, ktorý dokáže prejsť šachovnicu ľubovoľnej veľkosti väčšej alebo rovnej 5x5 (minimálny rozsah 5x5 až 20x20) ťahmi šachového koňa spôsobom, aby bolo každé políčko šachovnice navštívené práve raz, a zároveň poskytuje používateľovi možnosť zvoliť si ľubovoľnú začiatočnú pozíciu. Do programu je potrebné zapracovať takzvanú Warnnsdorfovu heuristiku, ktorej princíp spočíva v tom, že sa šachový kôň vždy posunie na políčko, odkiaľ má najmenej možností posunúť sa ďalej. V ďalšom kroku je potrebné nájsť pre 10 rôznych východzích pozícií na šachovnici rozmerov 8x8 jedno správne riešenie a program patrične zdokumentovať.

2 Navrhnuté riešenie

2.1 Použité programové prostriedky

Na riešenie úlohy bol použitý programovací jazyk C# v spojení s .NET framework verzie 6.0.300. Riešenie bolo vyvíjané v prostredí Visual Studio Code verzie 1.72.1 a pri jeho vývoji boli využité základné objektovo orientované princípy spolu s návrhovým vzorom Singleton.

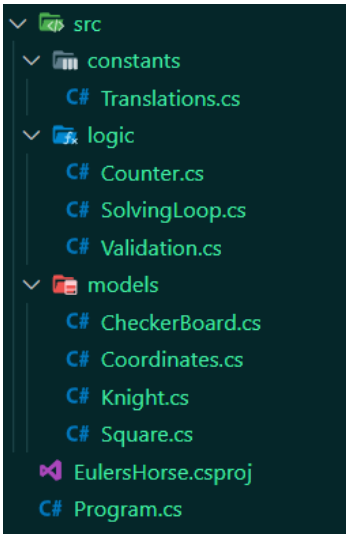
2.2 Organizácia projektu

Implementácia sa nachádza v adresári s názvom "*src*". Tento adresár obsahuje hlavné súbory *Program.cs* a *EulerHorse.csproj*, slúžiace na spustenie vykonávania programu, a trojicu ďalších adresárov *constants, logic* a *models*.

Adresár *constants* obsahuje jediný súbor *Translations.cs* so všetkými ôsmimi dovolenými krokmi šachovnicového koňa.

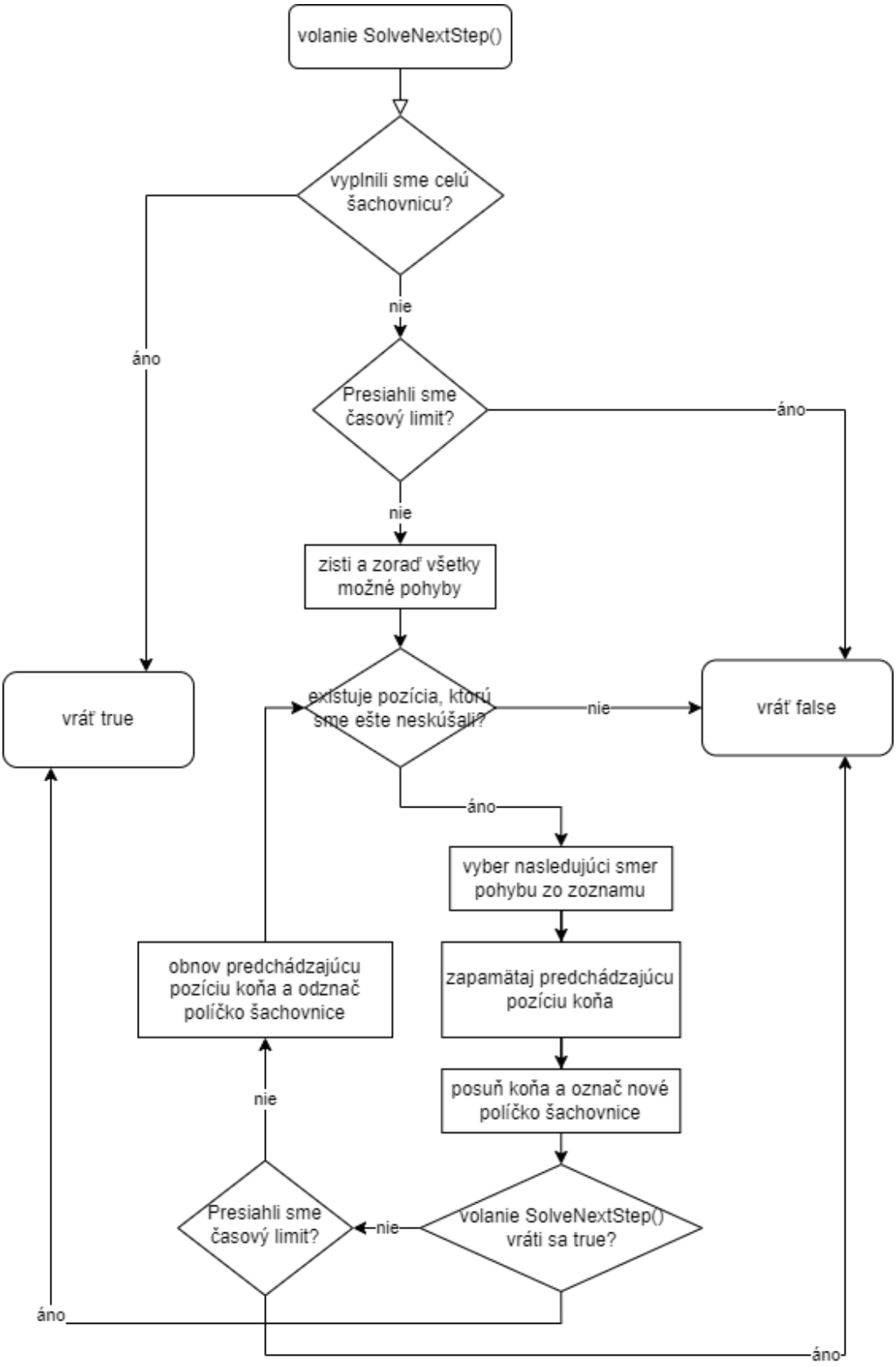
Adresár *models* obsahuje objektovú reprezentáciu hlavných prvkov programu. Nachádzajú sa tu triedy *CheckerBoard.cs* predstavujúci šachovnicu ako takú, *Square.cs* reprezentujúci jedno políčko šachovnice, *Knight.cs* predstavujúci šachovnicového koňa a nakoniec *Coordinates.cs*, z ktorého dedia *Knight.cs* aj *Square.cs* a ktorý obsahuje horizontálnu a vertikálnu súradnicu prvkov (či už statickú (nemeniacu sa) pozíciu štvorca na šachovnici alebo neustále sa meniacu pozíciu šachovnicového koňa).

Posledný a zároveň hlavný adresár *logic* obsahuje jadro programu spolu s pomocnými funkcionalitami. Súbor *SolvingLoop.cs* slúži na nájdenie riešenia problému s využitím Warnnsdorfovej heuristiky a rekurzie, statická trieda *Validation.cs* obsahuje metódy na validáciu pohybu šachovnicového koňa a posledná trieda *Counter.cs* predstavuje nástroj na meranie času vykonávania programu v milisekundách s presnosťou troch desatinných miest.



Obr. 2.1 Adresárová štruktúra implementácie

2.3 Diagram algoritmu prehľadávania



Obr. 2.2 Vývojový diagram rekurzívneho hľadania správnej cesty

2.4 Slovný opis algoritmu prehľadávania

Algoritmus je kombináciou prehľadávania do hĺbky a Warnnsdorfovej heuristiky, pričom ak existuje priama cesta od začiatku k cieľu (algoritmus nenatrafí na slepú uličku), tak sa vykoná iba počet krokov identický s počtom políčok šachovnice. Logika obsiahnutá v slučke rekurzívneho prehľadávania je relatívne jednoduchá. Na začiatku každého volania metódy *SolveNextStep()* sa zistí, či už nebola vyplnená posledná prázdna pozícia šachovnice a, ak ešte nebola, či už neuplynul časový limit 20 sekúnd.

```
// if we managed to fill in the last position, the tour is completed
if (currentValue == Board.NumOfSquares) {
    return true;
}

// if the solution is taking too long to compute, end execution
if (Counter.Get().IsOverLimit) {
    return false;
}
```

Obr. 2.3 Overenie, či má zmysel pokračovať v hľadaní (metóda SolveNextStep() triedy SolvingLoop)

V prípade, ak časový limit ešte neuplynul, v ďalšom kroku sa pomocou metódy *RankTranslations()* zvolia všetky ťahy, pomocou ktorých sa kôň dokáže presunúť z jeho súčasnej pozície na novú, ešte nenavštívenú pozíciu.

```
public List<(int, (int, int))> RankTranslations ((int xCoord, int yCoord) knightCoords)
{
    var rankedTranslations = new List<(int, (int, int))>();

    foreach ((int xCoord, int yCoord) translation in Translations.All())
    {
        int nextX = knightCoords.xCoord + translation.xCoord;
        int nextY = knightCoords.yCoord + translation.yCoord;

        if (Validation.ValidatePos(Board.Size, (nextX, nextY)) &&
            !Board.Squares[nextX, nextY].WasVisited) {
            var onwardMoves = Board.GetOnwardMovesFromSquare((nextX, nextY));

            rankedTranslations.Add((onwardMoves, translation));
        }
    }
    return rankedTranslations.OrderBy(tr => tr.Item1).ToList();
}
```

Obr. 2.4 Metóda RankTranslations() triedy SolvingLoop

V tejto metóde sa iteruje cez všetky existujúce kroky pohybu a pomocou metód z triedy *Validations.cs* sa zisťuje, či je možné daný krok vykonať. Ak ide o legálny krok, pomocou metódy *GetOnwardMovesFromSquare()* triedy *CheckerBoard.cs* sa začne uplatňovať Warnnsdorfova heuristika – táto metóda totižto zisťuje váhu kroku (t.j. počet legálnych krokov, ktoré sú z tejto novej pozície uskutočniteľné) a podľa tejto váhy sú jednotlivé legálny kroky v zozname zoradené vzostupne.

```
public int GetOnwardMovesFromSquare ((int xCoord, int yCoord) squareCoords)
{
    int numOfMoves = 0;

    foreach ((int xCoord, int yCoord) translation in Translations.All())
    {
        int nextX = squareCoords.xCoord + translation.xCoord;
        int nextY = squareCoords.yCoord + translation.yCoord;

        if (Validation.ValidatePos(Size, (nextX, nextY)) &&
            !Squares[nextX, nextY].WasVisited) {
            numOfMoves++;
        }
    }
    return numOfMoves;
}
```

Obr. 2.5 Metóda GetOnwardMovesFromSquare triedy CheckerBoard

Akonáhle má metóda *SolveNextStep()* k dispozícii všetky legálne kroky pohybu vo vzostupne zoradenom zozname, zvolí sa prvý krok (s najnižšou cenou), kôň sa pomocou tohto kroku presunie na novú pozíciu, pozícia sa označí (tieto údaje sú uložené v inštancii koňa, resp. štvorca šachovnice) a začne sa rekurzívne volanie *SolveNextStep()*, pričom toto volanie pokračuje buď dovtedy, kým nie je splnená jedna z podmienok na Obr. 2.3, alebo sa prehľadajú všetky možnosti a zistí sa, že pre daný vstup neexistuje riešenie.

```
    foreach (var tr in RankTranslations(Knight.GetCoords))
    {
        var newTranslation = tr.Item2;
        var previousPos = Knight.GetCoords;

        Knight.TranslateForward(newTranslation);
        Board.MarkSquare(++currentValue, Knight.GetCoords, previousPos);

        if (SolveNextStep()) {
            return true;
        }
        else if (Counter.Get().IsOverLimit) {
            break;
        }
        else {
            BackToPreviousSquare(Knight.GetCoords);
            Knight.TranslateBackward(newTranslation);
        }
    }
return false;
```

Obr. 2.6 Rekurzívna časť metódy SolveNextStep()

2.5 Možné výsledky prehľadávania

Počas vykonávania prehľadávania môžu nastať tri situácie, kvoli ktorým sa hľadanie ukončí. Najžiadúcejšia z nich nastane, ak je splnená prvá podmienka na Obr. 2.3, t.j. program našiel riešenie pre daný vstup a rekurzia sa ukončila. V tomto bode každé rekurzívne volanie

postupne vráti hodnotu “true”. Druhá situácia nastane, keď vypršal stanovený časový limit a program nestihol nájsť riešenie, prípadne riešenie neexistuje. Vtedy bude splnená druhá podmienka na Obr. 2.3 v poslednom rekurzívnom volaní a pri každom predchádzajucom bude splnená podmienka v “else if” vetve na Obr. 2.6, t.j. postupne sa bude vraciať hodnota “false”, až kým program neskončí. Posledná, tretia situácia sa stane realitou, keď metóda *RankTranslations()* vráti prázdny zoznam krokov, čo značí, že sa kôň ocitol v slepej uličke a nemá sa kam posunúť. V tomto bode sa vráti hodnota “false”, v predchádzajúcom rekurzívnom volaní sa vykoná vetva “else” na Obr. 2.6, odznačí sa daný štvorec šachovnice a kôň sa posunie späť na predchádzajúcu pozíciu. Keď sú tieto kroky vykonané, začne ďalšia iterácia v cykle foreach (z utriedeného zoznamu krokov sa vyberie nasledujúci krok) a hľadanie pokračuje ďalej dovtedy, kým sa cesta nenájde, nevyčerpajú sa všetky možnosti v každom rekurzívnom volaní alebo nevyprší časový limit.

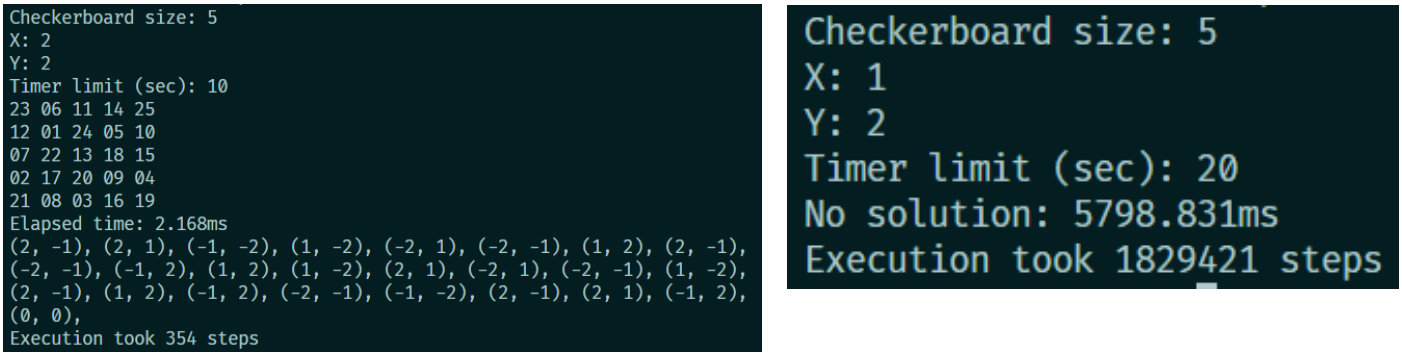
3 Testovanie správnosti

3.1 Spôsob testovania

Testovanie prebiehalo so šachovnicami rôznych veľkostí, primárne v rozmedzí od 5x5 do 35x35. Používateľské rozhranie programu umožňuje manuálne zadanie rozmeru šachovnice, súradníc začiatku a časového limitu (v sekundách) z konzoly a po nájdení správneho riešenia poskytne používateľovi vykreslenie vyplnenej šachovnice, postupnosť vykonaných operátorov, časový údaj – koľko milisekúnd trvalo nájsť riešenie a počet vykonaných krokov.

3.1.1 Testovanie šachovnice 5x5

Pri rozmeroch šachovnice 5x5 program stihne aj za relatívne krátky časový interval (do 5 sekúnd) prehľadať všetky možné cesty a za daný čas vykoná približne milión až dva milióny krokov (v prípade, ak riešenie neexistuje).



Obr. 2.7 Ukážka nájdenia riešenia pre rozmer 5x5 so vstupom (2,2) vľavo a nenájdenia riešenia (prehľadania všetkých možností) so vstupom (1,2) vpravo

Tabuľka nižšie obsahuje porovnanie času vykonávania a počtu vykonaných krokov v oboch prípustných konečných situáciách pre veľkosť 5x5.

5x5	začiatok	čas	kroky	výsledok
1	X=2, Y=4	0.881ms	25	priamo nájdený
2	X=2, Y=2	2.243ms	354	nájdený backtrackingom
3	X=2, Y=5	5682.733ms	1829421	nenájdený (neexistuje)

Tab. 2.1 Výsledky testovania šachovnice 5x5 pre všetky možné výsledné situácie

3.1.2 Testovanie šachovnice 8x8

Podľa zadania je potrebné pre šachovnicu rozmerov 8x8 nájsť 10 správnych riešení pre 10 rôznych začiatočných pozícií. Pri šachovniciach tejto veľkosti sa neobjavujú žiadne problémy – program počas testovania dokázal nájsť riešenie pre všetky zvolené začiatočné pozície, pričom ich našiel priamo (nikdy sa nedostal do slepej uličky).

8x8	začiatok	čas	kroky	výsledok
1	X=1, Y=1	0.52ms	64	priamo nájdený
2	X=7, Y=3	1.353ms	64	priamo nájdený
3	X=2, Y=3	0.684ms	64	priamo nájdený
4	X=5, Y=5	1.105ms	64	priamo nájdený
5	X=1, Y=8	0.708ms	64	priamo nájdený
6	X=6, Y=3	0.303ms	64	priamo nájdený
7	X=2, Y=5	1.107ms	64	priamo nájdený
8	X=3, Y=3	0.437ms	64	priamo nájdený
9	X=8, Y=7	0.965ms	64	priamo nájdený
10	X=5, Y=4	1.428ms	64	priamo nájdený

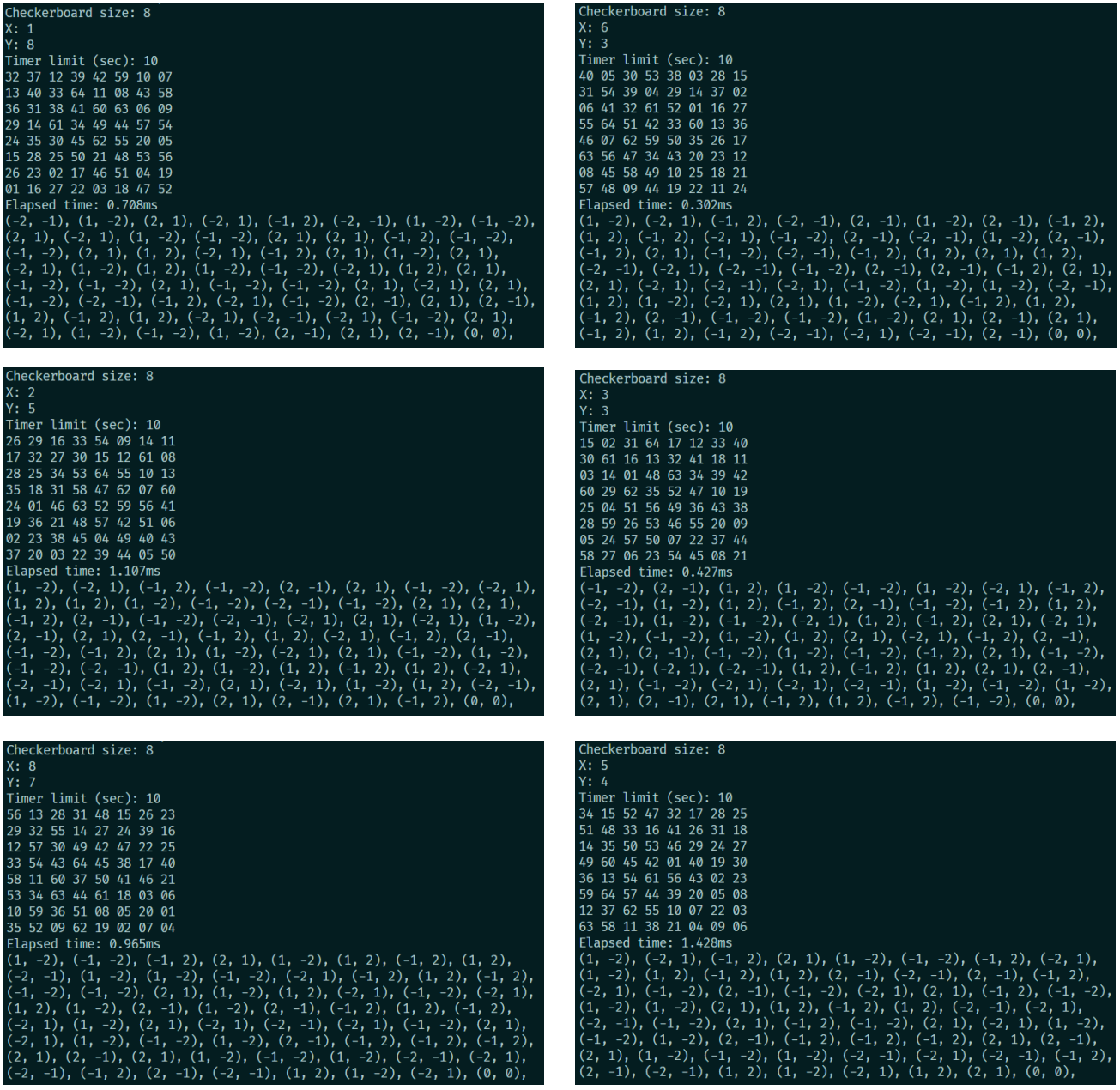
Tab. 2.2 Výsledky testovania šachovnice 8x8 pre 10 rôznych začiatočných pozícií

```
Checkerboard size: 8
X: 1
Y: 1
Timer limit (sec): 10
01 26 15 24 29 42 13 32
16 23 28 43 14 31 38 41
27 02 25 30 51 40 33 12
22 17 60 57 44 37 52 39
03 64 21 50 59 56 11 34
18 49 58 61 36 45 08 53
63 04 47 20 55 06 35 10
48 19 62 05 46 09 54 07
Elapsed time: 0.52ms
(1, -2), (-2, -1), (-1, 2), (1, 2), (-2, -1), (2, -1), (-1, 2), (-2, -1),
(1, -2), (-2, -1), (-1, 2), (1, 2), (2, 1), (1, -2), (2, -1), (1, -2),
(-2, 1), (-2, -1), (-1, 2), (1, 2), (1, 2), (-2, 1), (-2, -1), (2, -1),
(-2, -1), (1, 2), (1, -2), (1, -2), (-2, -1), (-1, 2), (1, 2), (-1, 2),
(2, -1), (2, -1), (-1, 2), (2, -1), (2, -1), (-1, 2), (-1, -2), (-1, 2),
(2, -1), (1, -2), (-2, -1), (-1, -2), (2, -1), (1, 2), (-1, 2), (1, 2),
(-2, 1), (-2, -1), (-2, 1), (-1, -2), (1, -2), (-1, -2), (2, -1), (-1, 2),
(-1, -2), (2, 1), (2, -1), (2, 1), (1, 2), (-1, 2), (1, 2), (0, 0),
```

```
Checkerboard size: 8
X: 7
Y: 3
Timer limit (sec): 10
63 58 07 34 21 32 05 02
08 35 64 57 06 03 20 31
59 62 47 22 33 56 01 04
36 09 60 53 48 23 30 19
61 46 49 40 55 42 15 24
10 37 54 43 52 27 18 29
45 50 39 12 41 16 25 14
38 11 44 51 26 13 28 17
Elapsed time: 1.353ms
(2, 1), (-1, -2), (1, -2), (-2, 1), (2, 1), (-1, 2), (-2, -1), (-2, -1),
(1, -2), (2, -1), (-1, 2), (-1, -2), (1, -2), (2, 1), (-1, 2), (-2, 1),
(2, 1), (1, -2), (1, -2), (-2, -1), (-1, 2), (-2, 1), (1, -2), (1, 2),
(1, -2), (2, -1), (-1, 2), (1, 2), (-1, 2), (-2, 1), (-1, -2), (-1, 2),
(-2, -1), (1, -2), (-1, -2), (1, -2), (1, 2), (1, -2), (-2, 1), (-1, 2),
(2, 1), (2, 1), (-1, 2), (-2, -1), (-1, -2), (1, -2), (-1, -2), (2, 1),
(-1, 2), (-1, -2), (2, -1), (2, 1), (2, -1), (1, 2), (-1, 2), (1, 2),
(-2, 1), (-2, -1), (-2, 1), (-1, -2), (2, 1), (-2, 1), (1, -2), (0, 0),
```

```
Checkerboard size: 8
X: 2
Y: 3
Timer limit (sec): 10
02 17 58 23 04 19 36 33
59 24 03 18 57 34 05 20
16 01 64 55 22 37 32 35
25 60 45 48 63 56 21 06
44 15 52 61 54 47 38 31
51 26 49 46 41 62 07 10
14 43 28 53 12 09 30 39
27 50 13 42 29 40 11 08
Elapsed time: 0.684ms
(-2, -1), (-1, -2), (2, 1), (2, 1), (1, 2), (-2, 1), (-2, -1), (-1, -2),
(2, 1), (-1, -2), (1, -2), (1, 2), (2, -1), (-1, -2), (-1, 2), (-1, 2),
(-2, -1), (2, -1), (1, 2), (2, -1), (-1, -2), (-2, -1), (-1, 2), (-1, -2),
(-2, 1), (1, 2), (1, 2), (-1, 2), (-1, -2), (2, 1), (-2, 1), (1, -2),
(-1, -2), (1, -2), (2, -1), (2, 1), (2, -1), (-1, 2), (1, 2), (-1, 2),
(-2, 1), (-1, -2), (-2, -1), (-1, 2), (2, 1), (2, -1), (2, 1), (1, -2),
(-1, -2), (1, -2), (-2, -1), (-2, 1), (-2, -1), (-1, 2), (2, -1), (-2, -1),
(1, 2), (-1, 2), (1, 2), (2, 1), (2, -1), (2, 1), (-1, -2), (0, 0),
```

```
Checkerboard size: 8
X: 5
Y: 5
Timer limit (sec): 10
43 12 61 56 41 14 37 48
64 53 42 13 60 49 40 15
11 44 57 62 55 38 47 36
52 63 54 45 50 59 16 39
25 10 51 58 01 46 35 30
04 07 24 27 34 31 20 17
09 26 05 02 19 22 29 32
06 03 08 23 28 33 18 21
Elapsed time: 1.105ms
(-1, -2), (-2, 1), (1, 2), (-2, -1), (-1, -2), (2, -1), (1, 2), (-1, 2),
(-1, -2), (2, -1), (1, 2), (1, -2), (-2, -1), (-2, 1), (-1, 2), (-2, 1),
(1, -2), (1, -2), (2, 1), (2, 1), (1, 2), (-2, -1), (-2, 1), (-2, -1),
(-1, -2), (2, 1), (-1, 2), (-1, -2), (1, -2), (2, -1), (-1, -2), (-2, 1),
(2, 1), (-2, 1), (1, -2), (2, -1), (1, 2), (2, -1), (1, 2), (-2, -1),
(-1, -2), (-2, 1), (-2, -1), (1, 2), (2, -1), (-2, -1), (-1, 2), (1, 2),
(-1, 2), (2, 1), (2, -1), (2, 1), (1, -2), (-1, -2), (1, -2), (-2, -1),
(1, 2), (1, -2), (-2, 1), (2, 1), (-1, -2), (-2, 1), (-1, 2), (0, 0),
```



Obr. 2.8 Výstup programu pre všetkých 10 nájdených riešení pre veľkosť 8x8

3.1.3 Testovanie šachovnice väčších rozmerov

Pri šachovniciach väčších ako 7x7 sa dostávame do pozície, kedy v prípade, ak riešenie neexistuje, program väčšinou nestihne zkontrolovať všetky možné kombinácie a skončí sa kvôli uplynutiu časového limitu. Na druhej strane, ak existuje priame riešenie bez slepých uličiek, Warnnsdorfova heuristika zaručí, že ho program nájde za skoro identicky krátky čas, ako pri šachovniciach menších rozmerov. Pri každom vykonanom teste bol použitý časový limit 20 sekúnd, po uplynutí ktorého program hlásil neúspešné hľadanie.

rozmer	začiatok	čas	kroky	výsledok
7x7	X=5, Y=3	1883.289ms	542762	nájdený backtrackingom
10x10	X=2, Y=6	1.269ms	100	priamo nájdený
15x15	X=15, Y=3	2.216ms	225	priamo nájdený
20x20	X=17, Y=12	2.805ms	400	priamo nájdený
25x25	X=21, Y=7	4.178ms	625	priamo nájdený
25x25	X=24, Y=7	limit 20s	4894639	nenájdený (neexistuje)
35x35	X=17, Y=31	7.917ms	1225	priamo nájdený
50x50	X=44 Y=28	17.196ms	2500	priamo nájdený

Tab. 2.3 Výsledky testovania šachovníc rôznych väčších rozmerov

3.2 Zhodnotenie testovania

Z nazbieraných údajov uvedených v predchádzajúcej podkapitole môžeme dospieť k niekoľkým zisteniam. Počas testovania nenastal prípad, kedy by pri šachovnici párnej veľkosti (napr. 6x6, 8x8) program nedokázal nájsť aspoň jedno správne riešenie, a rovnako ani situácia, kedy by sa šachovnicový kôň ocitol v slepej uličke a musel by sa vrátiť na predchádzajúci štvorec. Naopak, zistila sa zaujímavá okolnosť. Pri šachovniciach nepárnej veľkosti (napr. 9x9, 11x11) a začiatočnej pozícii, ktorá obsahuje jednu párnu a jednu nepárnu súradnicu (napr. X=7, Y=6) nikdy neexistuje riešenie. Tento postreh sa overoval na šachovnici veľkostiach 5x5, kde program stihne prekontrolovať všetky možné kombinácie pohybov za dobu menej ako 10 sekúnd.

Čas potrebný na vyriešenie problému spĺňa očakávania. Ak existuje priama cesta od začiatočnej pozície k cieľovej, program ju nájde za veľmi krátky čas, rádovo niekoľko milisekúnd. Pamäťová zložitosť je tiež veľmi priaznivá – počas behu programu sa nevytvárajú žiadne nové objekty (políčka šachovnice sa vytvoria ihneď na začiatku programu) a pohyb šachovnicového koňa je zaznamenaný iba formou dvojíc súradníc (X, Y).

3.3 Možnosti rozšírenia a optimalizácie riešenia

Jeden z možných spôsobov rozšírenia programu sa týka zmeny algoritmu, akým sa zoradia legálne kroky v metóde *RankTranslations()*. V súčasnej implementácii nie je ošetrená situácia, kedy viacero štvorcov má rovnakú cenu (v tom prípade sa prvý posun zvolí náhodne). Pre zvýšenie efektivity a algoritmu by sa v takejto situácii mohla určiť vzdialenosť týchto štvorcov od okrajov šachovnice a následne by boli zoradené podľa tohto kritéria.

Jednoduchý spôsob, ako by sa dalo vyhľadávanie viac optimalizovať, je automatické vyhodnocovanie situácií, kde vopred vieme povedať, že správne riešenie neexistuje. Medzi takéto situácie patrí aj okolnosť opísaná v predchádzajúcej podkapitole so šachovnicami nepárnej veľkosti.