Zadanie 2 - Eulerov kôň

Umelá inteligencia

Pavol Krajkovič

2020/2021

Opis problému – zadanie

1. Opis problému

Eulerov kôň

Úlohou je prejsť šachovnicu legálnymi ťahmi šachového koňa tak, aby každé políčko šachovnice bolo prejdené (navštívené) práve raz. Riešenie treba navrhnúť tak, aby bolo možné problém riešiť pre štvorcové šachovnice rôznych veľkostí (minimálne od veľkosti 5 x 5 do 20 x 20) a aby cestu po šachovnici bolo možné začať na ľubovoľnom východziom políčku.

Jedno z mnohých riešení Eulerovho koňa s šachovnicou o rozmeroch 5x5 je napríklad toto:

1	16	21	10	7
22	11	8	15	20
17	2	25	6	9
12	23	4	19	14
3	18	13	24	5

2. Zadanie

g)

Problém 3. Pre riešenie problému Eulerovho koňa existuje veľmi dobrá a pritom jednoduchá heuristika, skúste na ňu prísť sami. Ak sa vám to do týždňa nepodarí, pohľadajte na dostupných informačných zdrojoch heuristiku (z roku 1823!), prípadne konzultujte na najbližšom cvičení cvičiaceho. Implementujte túto heuristiku do algoritmu prehľadávania stromu do hĺbky a pre šachovnicu 8x8 nájdite pre 10 rôznych východzích bodov jedno (prvé) správne riešenie (pre každý východzí bod). Algoritmus s heuristikou treba navrhnúť a implementovať tak, aby bol spustiteľný aj pre šachovnice iných rozmerov než 8x8. Treba pritom zohľadniť upozornenie v Poznámke 1. Je preto odporúčané otestovať implementovaný algoritmus aj na šachovnici rozmerov 7x7, 9x9, prípadne 20x20 (máme úspešne odskúšaný aj rozmer 255x255) a prípadné zistené rozdiely v úspešnosti heuristiky analyzovať a diskutovať.

3. Poznámka

Pre problém Eulerovho kôňa treba v oboch úlohách uvažovať s tým, že pre niektoré východzie políčka a niektoré veľkosti šachovnice riešenie neexistuje. Program preto treba navrhnúť a implementovať tak, aby sa v prípade, že do určitého času, resp. počtu krokov riešenie nenájde, zastavil a signalizoval neúspešné hľadanie. Maximálny počet krokov, resp. maximálny čas hľadania by preto mal byť ako jeden zo vstupných (voliteľných) parametrov programu. Pre toto zadanie a testovacie príklady je odporúčaný maximálny počet krokov jeden až desať miliónov, resp. maximálny čas 15 sekúnd.

Implementované spôsoby riešenia

Pre riešenie tohto zadania som sa rozhodol implementovať viacej techník ako dostať cestu eulerovho koňa. Každá metóda riešenia pracuje na inej báze a tak môžeme dostať rôzne výsledky ohladom času, počtu stavov a nájdením eulerovej cesty ako samotnej. Algoritmy budú testované na tej istej vzorke. Rozhodol som sa spracovať zadanie v Pythone.

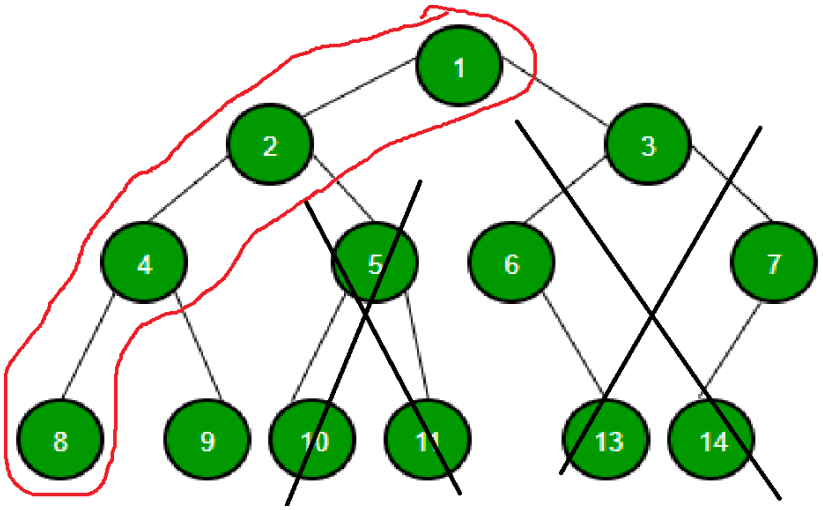
Eulerovu cestu vieme nájsť na šachovnici cez obyčajné ***prehľadávnie stromu do hĺbky***. Pri šachovniciach veľkosti n >=7 je počet vetiev obrovský a cesta sa nenájde. Preto potrebujeme zaviesť nejakú taktiku, aby program vedel zhruba kde v strome sa nachádza riešenie na náš problem.

Čisté Warnsdorffove pravidlo

A teda ako prvú techniku som sa rozhodol implementovať čistú heuristiku, ktorá nám bola poskytnutá v zadaní a to ***Warnsdorffove pravidlo (1823)***. Je to heuristika pre nájdenie jednej cesty eulorovho koňa. Pracuje na pravidlu, že sa kôň bude pohybovať na políčko, z ktorého sa kôň može dostať na čo najmenej iných políčok. Problém pri použití čistej heuristiky je, že počas behu programu sa môžeme stretnúť s dvoma a viac bodmi, ktorý majú taký istý stupeň, teda toľko istých susedov na ktorých sa môže kôň dostať. Algoritmus si vyberie ten bod,ktorý je na začiatku poľa a druhý neberie do úvahy, aj keď môže byť viac optimálny neskôr (resp. cezeň by cestu našiel). Výhodou tejto metody je, že program skončí po **n = veľkost_šachovnice ** 2 krokoch.** Druhá a asi najvacsia výhoda je, že vie nájsť cestu pre každú šachovnicu rôznych rozmerov ak dáme štartovacie políčko také, kde počas cesty nemusíme prehladavať viac stavov v rovnakej hlbke, keďže backtracking som do prvej metody nezaimplementoval. A preto **nás tu neobmedzuje veľkosť pamate a ani max hĺbka rekurzie**. Nevýhodou je ,že prehladáva iba jednu cestu v strome a ak na konci narazí na dead end a šachovnica nebola cela prejdena tak sa nevráti naspäť aby vyskúšal ine vetvy (znovu, heuristika bez backtrackingu).

ALGORITMUS:

1. Nastav číslo tahu pre štartovaci bod
2. Prehladaj susedov bodu
3. Zoraď vzostupne susedov bodu poďla stupňa
4. Zober suseda na prvom indexe, nastav mu čislo tahu +1 a pokračuj na krok 2
5. Vrát true ak sa našla cesta, inak false

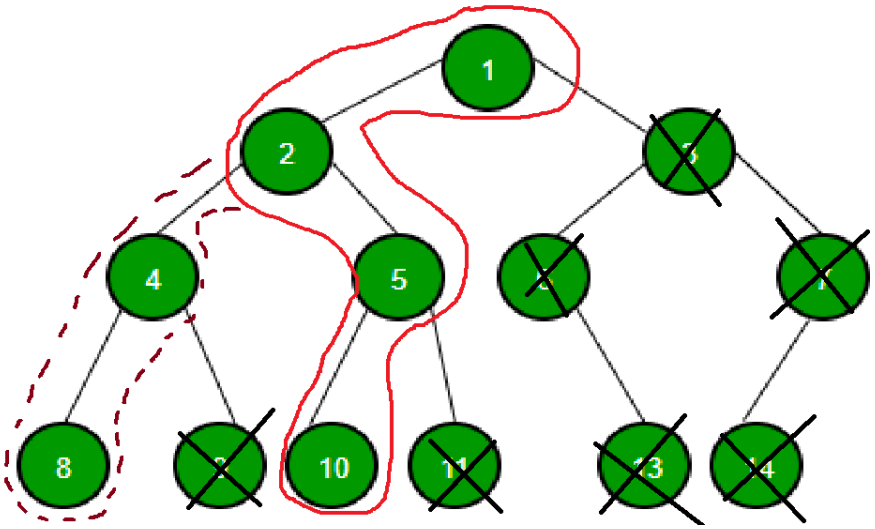


Warnsdorff + Backtracking

Ako druhú metódu som použil tiež Warnsdorffove pravidlo ale je vnorené do prehľadávania do hĺbky, **teda Warnsodorffove pravidlo + backtracking**. Algoritmus nájde cestu ak nejaká existuje v danom strome. Pokračuje podľa Warsndorffoveho pravidla pokiaľ môže. Ak narazí na koniec, s tým, že ešte neprehladal celý strom, tak sa vráti na predchádzajúci uzol a znovu sa začne správať podľa pravidla. Táto metóda je náročnejšia na pamäť a čas, keďže je spravená rekurzívne a v jednom čase sa ukladá veľa stavov. Výhoda tejto metódy je, že problém ktorý nastal v minulej technike, že sme si nevedeli vybrať medzi dvomi rovnocennými bodmi, tu nenastane. Ak nenájde algoritmus cestu, tak sa prehľadá ako prvý tak aj druhý bod. Nevyhodou na druhej strane je, že je táto metóda je náročnejšia na pamäť, trvá dlhšie, keďže sa preskúmava viac možnosti.

ALGORITMUS:

1. Nastav číslo tahu pre štartovaci bod
2. Prehladaj susedov bodu
3. Zoraď vzostupne susedov bodu poďla stupňa
4. Zober suseda na prvom indexe, nastav mu čislo tahu +1 a pokračuj na krok 2
5. Ak si na konci a nenašla sa cesta, zober bod zo zoznamu na indexe + 1
6. Vrát true ak sa našla cesta, inak false



Vlastná heuristika

Tretiu metódu som si vymyslel vlastnú (neefektívnu) heuristiku. Moja heuristika pracuje na takej báze, že kôň sa posunie na takého suseda, ktorý je čo najbližšie k stene šachovnice. Tým, že to nie je žiadna overená heuristika a nie je velmi efektivna, tak som ju automaticky implementoval do backtrackingu aby sa našla cesta ak sa nejaká zo štartovacieho políčka nachádza. Nevýhodou je, že je veĺmi drahá na pamäť, na čas a pri šachovniciach veľkej veľkosti mi pretečie pamäť a nič sa nevypíše. Výhodou avšak je, že som ju vedel využiť v špecialnom pripade pri poslednej techinke na eulerovu cestu a vďaka nej som dostal ešte lepšie výsledky ako čista Warnsdorffova heuristika + backtracking.

ALGORITMUS:

1. Nastav číslo tahu pre štartovaci bod
2. Prehladaj susedov bodu
3. Zoraď vzostupne susedov bodu poďla stupňa (najmensia vzdialenost od kraja)
4. Zober suseda na prvom indexe, nastav mu čislo tahu +1 a pokračuj na krok 2
5. Ak si na konci a nenašla sa cesta, zober bod zo zoznamu na indexe + 1
6. Vrát true ak sa našla cesta, inak false

Warnsdorffove pravidlo + backtracking + moja heuristika pri tiebreak

Ako bolo spomínané na začiatku, môžeme sa stretnúť s dvomi bodmi rovnakeho stupňa. Pri takomto strete prechádzame body tak, ako boli pridané do poľa susedov. Pri najhoršom prípade sa vieme stretnúť so siedmimi susedmi s rovnakým stupňom a ak je eulerova cesta až cez posledný bod, tak budeme musieť prejsť celým zoznamom susedov. Zaujímalo ma či vieme tento prípad nejako vylepšiť. Preto som sa rozhodol použíť moju heuristiku pri taktomto strete a preusporiadať body s rovnakými stupňami v zozname susedov pomocou mojej heuristiky. Prišlo mi logické, že body ktoré sú bližšie ku kraju šachovnici majú menšiu šancu byť navštívené v budúcnosti ako body, koré sú viac v strede šachovnice. Takže v zozname budú susedia ktorý sú sortnutý na báze dvoch kľúčov. Prvý krát podľa Warnsdorffoveho pravidla a druhýkrát poďla mojej heuristiky. Výhody si ukážeme v testovani, pre niektoré body ktoré sme nenašli na prvý krát (počet krokov == VELKOST_SACHOVNICE**2) cestu pri Warnsdorff + backtracking sa cesta našla takto na prvý krát.

1. Nastav číslo tahu pre štartovaci bod
2. Prehladaj susedov bodu
3. Zoraď vzostupne susedov bodu poďla prveho stupňa
4. Zober suseda na prvom indexe, nastav mu čislo tahu +1
5. Pozri či nie je viac uzlov s takým istým stupnom, ak je tak tieto uzly zorad aj podla druheho stupna (vlastnej heuristiky)

6. Pokračuj na krok 2
7. Ak si na konci a nenašla sa cesta, zober bod zo zoznamu na indexe + 1
8. Vrát true ak sa našla cesta, inak false

Používateľské rozhranie

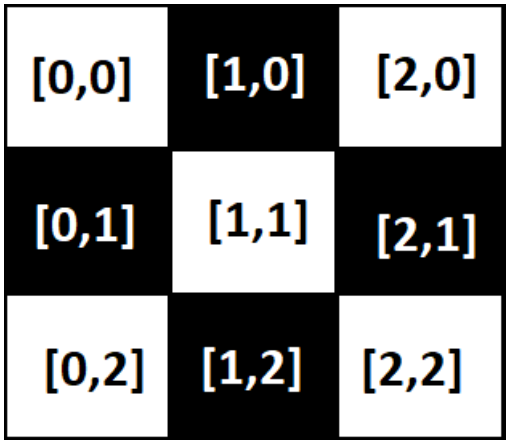
```
Zadaj velkost sachovnice: >? 8
[(5, 7), (3, 3), (0, 6), (2, 7), (1, 7), (2, 0), (1, 5), (0, 5), (0, 3), (5, 6)]
0 - Test 10 vychodzich random bodov
1 - Dostupnost cesty pre kazde policko na sachovnici pouzitim cistej heuristiky
2 - (BOD) Cesta pre cistu Heuristiku
3 - (BOD) Cesta pre vlastnu heuristiku
4 - (BOD) Cesta pre Heuristiku + backtracking
5 - (BOD) Cesta pre Heuristiku + backtracking + vlastná heuristika
Zadaj moznost:
```

Na začiatku musí uživatel zadať veľkost šachovnice, potom si vyberá možnosť. 0 predstavuje riešenie zadania a to nájdenie cesty – ak je to možné – pre 10 náhodných bodov na šachovnici. 1 – vypísanie pre ktoré body nájde čistá heuristika eulerov ťah koňa. Vybral som túto metódu iba lebo je z pomedzi heuristik najslabšia a ostatné, okem mojej, budujú na nej. Je veľmi rýchla a tak máme výsledok skoro ihneď. Od 2 – 5 sú techniky nájdenia eulerovej cesty koňa , ktoré som implementoval.

Reprezentácia údajov

Kôň má 8 možností v akom smere posunie, tieto ťahy mám uložené v array of tuples. Samozrejme môže sa stať, že kôň sa rozhodne výjsť zo šachovnice, to mám ošetrené cez funkciu **check_bounds**.

Šachovnicu reprezentujem cez 2D pole a ak chceme dostať cestu pre Eulerovho kona, tak k bodom tiež pristupujem ako cez indexy 2D pola. Pre šachovnicu 3x3 to vyzerá nasledovne



Body, ktoré posúvam do rekurzie ukladám do objektu CELL alebo sú uložené v zozname susedov vyzerá takto

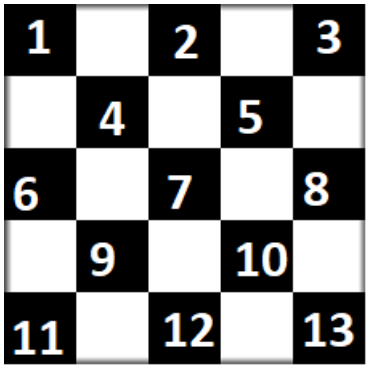
```
class cell:
    def __init__(self, x, y, move_number, degree):
        self.x = x
        self.y = y
        self.move_number = move_number
        self.degree = degree                # stupen ktory nam pomaha pri Warndorffovom pravidle, pocet nenavstivenych susedov
        self.secondary_degree = 2000        # stupen ktory nam pomaha pri mojej heuristike, vzdialenost od okraja sachovnice

    def add_secondary_degree(self, num):
        self.secondary_degree = num
```

Testovanie

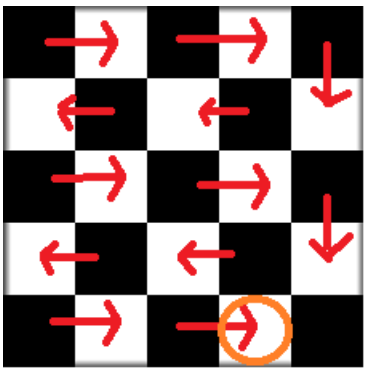
Pre testovanie zadania som spravil vlastnú možnosť v konzole a to je možnosť 0, ktorá mi vyhodnotí algoritmy pre 10 náhodných bodov. Vlastné testovanie je rozdelené do viac častí a tak vypíšem každú z nich + čo som sa dozvedel z nich. Stavy počítam od druhého políčka keďže prvé je už zapísane v 2D poli, ktoré po každom teste prečistujem.

A) Šachovnice s veľkosťou nepárneho čísla (7x7,5x5,9x9)

Kôň každým svojim krokom zmení farbu políčka. Z bielej ide na čiernu a z čiernej ide na bielu. Ak si zoberieme šachovnicu s veĺkostou párneho čísla tak je počet bielych aj čiernych políčok rovnaký. Naopak pri nepárnej veľkosti je počet jednej skupiny políčok o jeden viac. Zoberme si teda šachovnicu 5x5. Počet čiernych (políčok farby rohu) políčok je o jedna viac.



Ak vieme, že kôň pri každom tahu mení farbu políčka tak si pome popárovať biele a čierne políčko čo nám bude predstavovať ťah koňa. Samozrejme kôň sa nehýbe o jeden všetkým smerom ale hýbe sa v tvare L. To nás teraz nezaujíma.



Ak začíname na čiernom políčku v lavo hore a postupne striedame farby tak končíme na čiernom políčku v pravom dolnom rohu. Z predposledného políčka označenym oranžovým krúžkom sa vieme dostať na čierne políčko. Takže ak začneme v šachovniciach s nepárnou dĺžkou tak ak začneme na čiernom políčku, tak skončíme na čiernom políčku (bielych je o jedna menej a tak posledné čierne políčko nemá kamarata). Pri šachovniciach párnej dĺžky začneme na políčku bielom/čiernom a končíme na políčku opačnej farby ako bol začiatok.

Ak by sme ale začali na políčku bielej farby, tak pre šachovnicu 5x5 by bol 24 ťah na čierne políčko a tým, že je čiernych políčok o jedna viac tak by posledné ešte nenavštívene políčko bolo tiež čierne. No avšak vieme, že kôň mení farby každým ťahom a tym pádom ťah koňa z čierneho políčka na čierne neexistuje a teda cesta sa nenájde

Zhrnutie bodu A: Cesta pri nepárnej šachovnici z políčka ineho ako je farba rohu neexistuje

B) Problémový bod

Pomocou čistej heuristiky som na šachovnici 8x8 nenašiel na prvý krát cestu na dvoch bodoch. Preto som sa rozhodol ich analyzovať podrobnejšie. Môžem lepšie ukázať výhody a nevýhody algoritmov. Body to sú [0,5] a [1,3]. Rozhodol som sa analyzovať bod [0,5]

Čistá heuristika

```
[0][5]: not complete
Algoritmus trval 0.0010008811950683594 sekund
Pocet navstivenych stavov je :  63
 14   29   10   53   24   27   8    39
 11   58   13   28   9    40   23   26
 30   15   54   59   52   25   38   7
 55   12   57   48   41   60   45   22
 16   31   62   51   46   49   6    37
 1    56   47   34   61   42   21   44
 32   17   -1   3    50   19   36   5
 -1   2    33   18   35   4    43   20
```

Čistá heuristika alebo teda Warnsdorffovo pravidlo je samotný základ. Tým, že ide iba cez NxN počet stavov (kde N je veĺkosť šachovnice) tak ak nenašiel cestu na prvý krát tak sa vráti a vypíše, že nenašiel cestu. V tomto prípade cestu teda nenašiel, zostali mu dva body na ktoré sa kôň už nedostane

Moja Heuristika

```
[0][5] Nepodarilo sa najst cestu
Pocet navstivenych stavov je :  2306398
Algoritmus trval 15.000067234039307 sekund
```

Mojej heuristike sa taktiež nepodarilo nájsť správnu cestu, počet stavov bol rapídne väčší, program skončil na limit času, prekročil 15 sekund s tým, že nenašiel riešenie. Bolo to očakávané, lebo vlastná heuristika pracuje do šachovnice velkosti 7x7

Warnsdorff + backtracking

```
[0][5] Podarilo sa najst cestu
 14   29   10   53   24   27   8    39
 11   56   13   28   9    40   23   26
 30   15   54   63   52   25   38   7
 55   12   57   48   41   62   45   22
 16   31   64   51   46   49   6    37
 1    58   47   34   61   42   21   44
 32   17   60   3    50   19   36   5
 59   2    33   18   35   4    43   20

Pocet navstivenych stavov je :  90
Algoritmus trval 0.0009984970092773438 sekund
```

Warnsdorff + backtracking je prvá metóda, ktorá našla riešenie, na počte stavov môžeme vidieť, že na prvý krát nenašla riešenie tak sa musela vrátiť späť a hladať riešenie cez inú vetvu. Po porovnaní s výsledkom po čistej heuristike môžeme vidieť, že cesta sa rozdelila v 56 kroku. Kde čista heuristika vybrala 56 krok na políčko [1,5] zatiaĺ čo Warnsdorff + backtracking na políčko [1,1].

Warnsdorff + backtracking + moja heuristika pri tiebreak

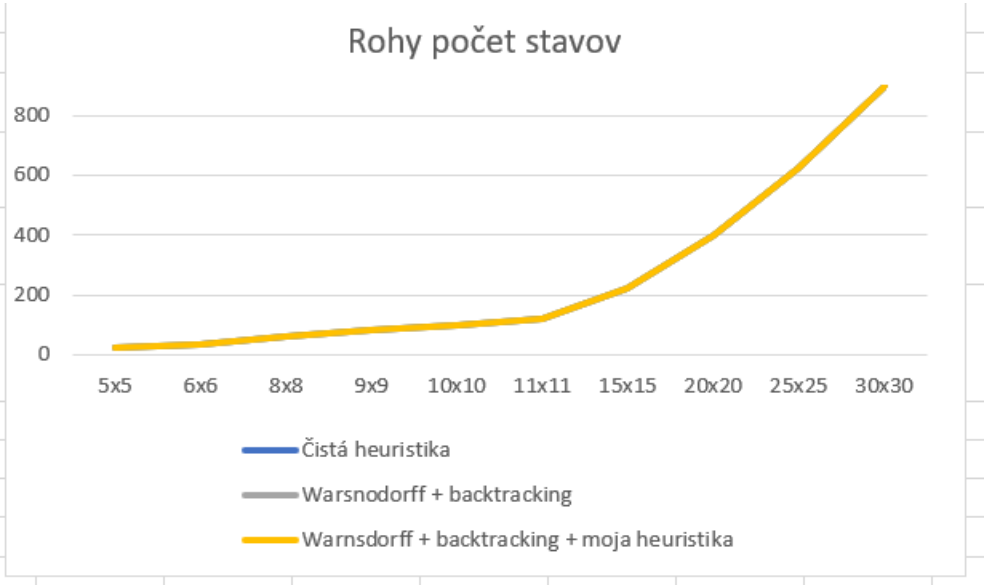
```
[0][5] Podarilo sa najst cestu
 14   29   10   49   24   27   8    51
 11   48   13   28   9    50   23   26
 30   15   64   45   54   25   52   7
 47   12   55   40   61   44   37   22
 16   31   46   63   38   53   6    43
 1    56   39   60   41   62   21   36
 32   17   58   3    34   19   42   5
 57   2    33   18   59   4    35   20

Pocet navstivenych stavov je :  63
Algoritmus trval 0.0010001659393310547 sekund
```

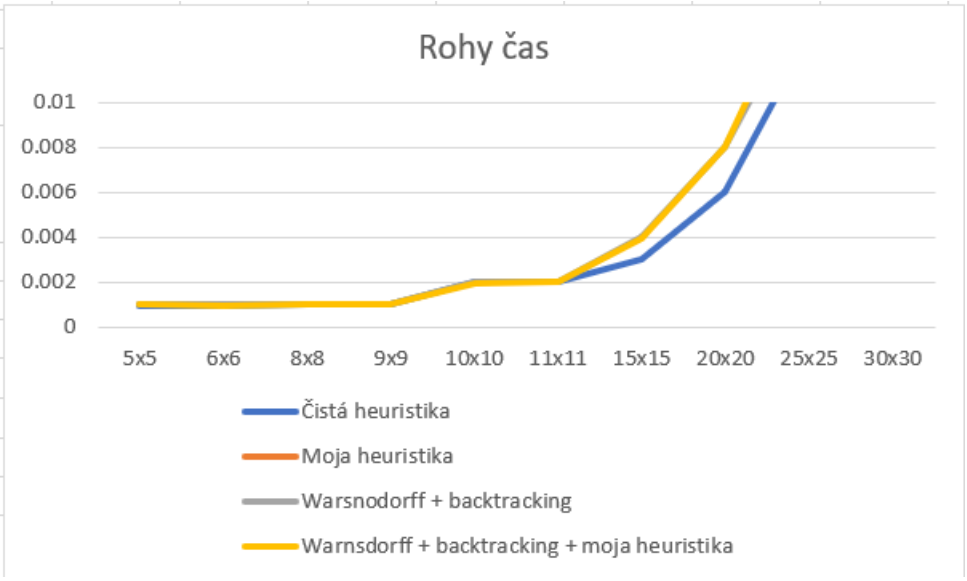
Najlepší výsledok sme dostali najzložitejším algoritmom, a to je môj mix vlastnej heuristiky s Warndorffovim pravidlom a backtrackingom. Môžeme vidieť, že tento algoritmus našiel cestu na prvý krát. V porovnaní s predošlou metodou sa rozide cesta pri kroku 34. Tento algortimus nám dal najlepšie výsledky. Dokonca našiel cestu na prvý krát.

Zhrnutie bodu B: Tým,že pri poslednej metode nemusíme slepo vyberať zo zoznamu susedov ak je ich tam viac s rovnakým stupnom, tak ich stačí este nejako zorganizovať, čo v tomto prípade bola moja heuristika. Ak by sme ju nezorganizovali ako v prípade Warnsdorff + backtracking tak iba slepo prechádzame celý zoznam.

Su tri druhy bodov s ktorými sa vieme stretnúť Rohove, Krajove a stredove, pome sa pozrieť na všetky. Tým, že data sú veĺke budem ich spracovávať v exceli. Dáta v exceli predstavujú priemerné data počas 5 testov pre každy bod.

C) Rohy šachovnice

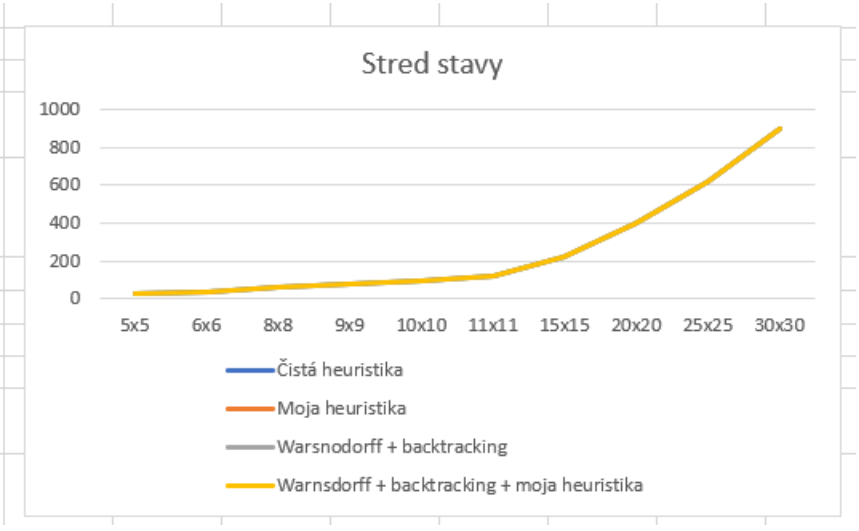


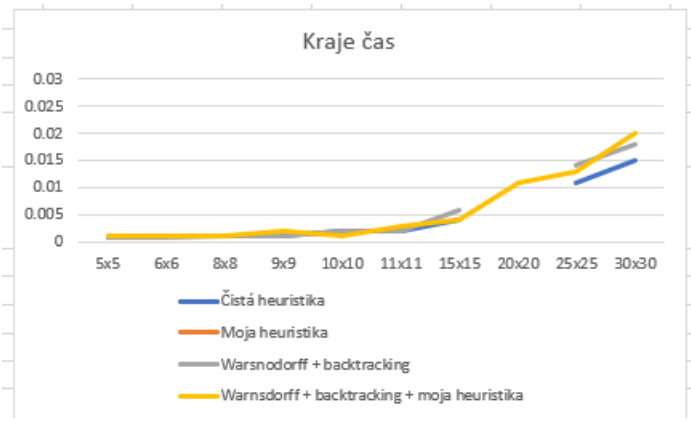


Zhrnutie bodu C: zistil som, že z rohovych bodov dokáže násjť cestu pre testované šachovnice aj obyčajný čisty Warnsdorff na prvý krát. Čo sa týka času, čistá heuristika je pri väčších šachovniciach rýchlejšia a to je preto, že nejde rekurzívne, nepotrebuje sa toľko vnárať.

D) Kraje šachovnice

Kraj šachovnice som vyberal bod [0,2]

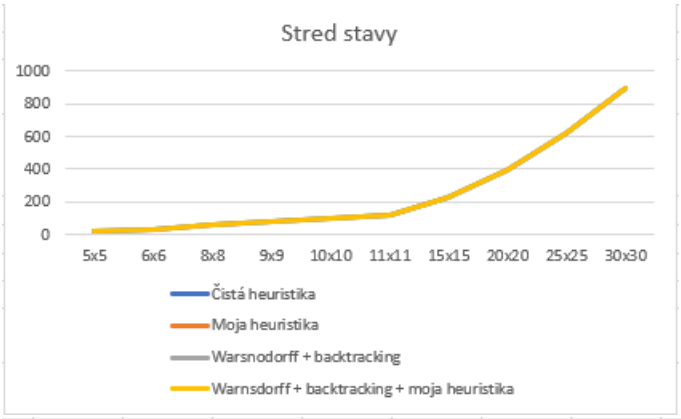


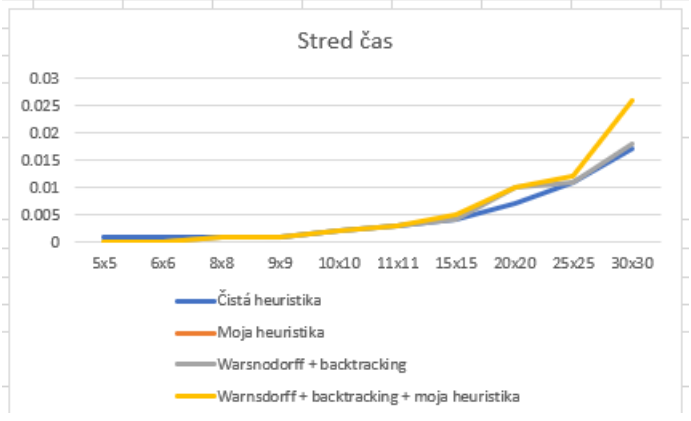


<u>Zhrnutie bodu D</u> posledný algoritmus Warnsdorff + backtracking + moj je najpomalší ku koncu ale dokáže nájsť cestu aj ked to ostatne nedokazu

E) Stred šachovnice

Stred som vyberal číslo velkosti šachovnice celočiselne delenie 2





Zhodnotenie riešenia

Testovanie

Pri testovani sa mi potvrdilo, že môj algoritmus nie je efektivny, prehladava velmi vela stavov a trva dlho. Zo všetkých trval najdlhšie posledný algoritmus a najrýchlejšie čista heuristika.

Warnsdorffove pravidlo + backtracking + moja heuristika pri tiebreak

Výhody – najrýchlejší, pre nájdenie cesty prejde najmenej stavov

Nevýhody – Drahý na pamäť, pri šachovniciach velkeho rozmeru pretečie pameť

Vlastná heuristika

Výhody – žiadne

Nevýhody – Drahý na pamäť, prejde zo všektych algoritmov najviac stavov, už po šachovnici 7x7 nenájde riešenie

Warnsdorffove pravidlo + backtracking

Výhody – Rýchli, pre nájdenie cesty prejde menej stavov ako čistá heuristika,

Nevýhody – Drahý na pamäť

Čisté Warnsdorffove pravidlo

Výhody – vždy prejde NxN stavov, je použitelny aj pri obrovských šachovniciach, lacný na pamäť

Nevýhody – nenájde cestu ak treba prehladať viac stavov v tej istej hĺbke