Umelá inteligencia

Zadanie č. 1

Eulerov kôň (Knight´s tour)

Norbert Matuška
9-25-2023

Norbert Matuška
UI Streda 13:00-14:40

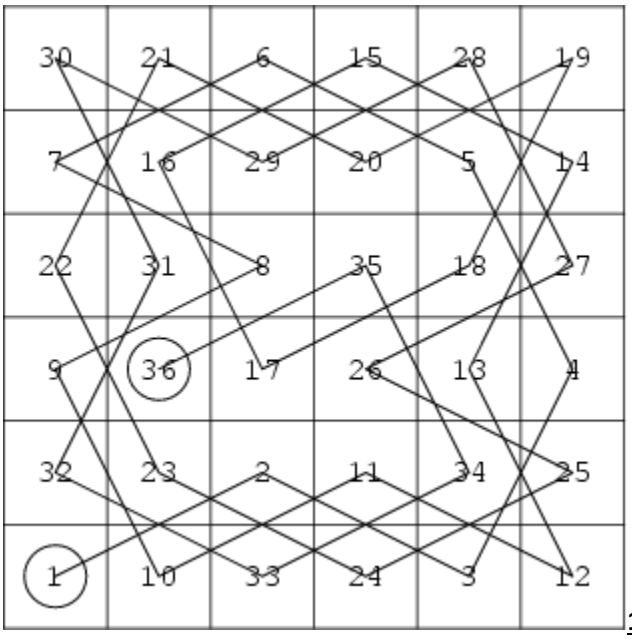
Contents

Norbert Matuška
UI Streda 13:00-14:40

Opis problému

Úlohou je prejsť šachovnicu rôznej veľkosti legálnymi ťahmi šachového koňa tak, aby sme navštívili každé jedno políčko na šachovnici práve jeden krát.

http://www2.fiit.stuba.sk/~kapustik/z2d.html#F



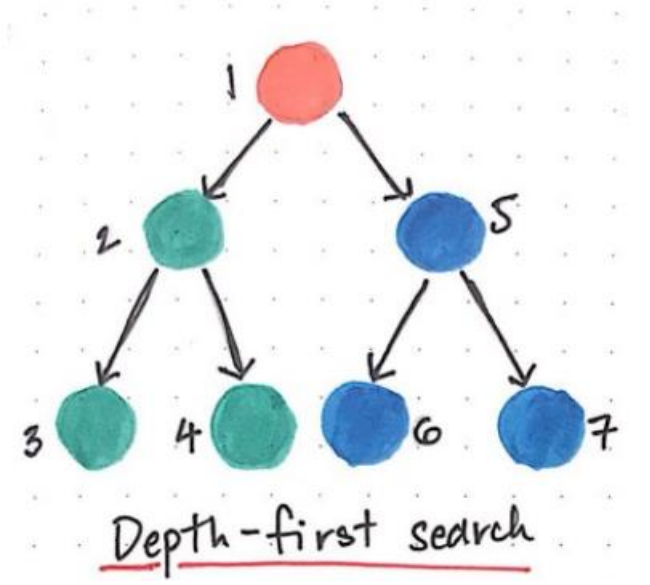
1.

Norbert Matuška
UI Streda 13:00-14:40

Algoritmus použitý na riešenie problému

Na riešenie problému som mal v zadaní použiť algoritmus slepého prehľadávania (Prehľadávanie do hĺbky, ďalej len DFS). DFS algoritmus sa snaží prehľadávať čo najhlbšie ako sa dá. Čiže hľadám cestu až kým nedôjdem na krok, v ktorom sa už nedá nikde posunúť bez toho aby som znova navštívil políčko alebo vystúpil von zo šachovnice. V takom prípade sa vrátim naspäť na posledný rozvinutý uzol a prehľadávam možnosti, ktoré ešte neboli navštívené, pokým nenájdem prvé riešenie kedy vráti cestu. Hľadanie do hĺbky nezaručuje nájdenie riešenia a taktiež nezaručuje nájdenie "najlepšej" cesty.

Diagram pre DFS



2.

Zložitosť algoritmu

Pamäťová zložitosť:

-je lineárna: $m * b$ kde m = maximálna hĺbka a b = faktor vetvenia

Časová zložitosť:

-$O(b^m)$ v najhoršom možnom prípade, kedy budeme musieť prezrieť všetky uzly, v najlepšom prípade sa môže nájsť riešenie hneď

Prípustnosť:

-DFS nezaručuje nájdenie najlepšej cesty/riešenia. Je šanca, že sa neoptimálne riešenie nájde skôr ako to najlepšie. Takže tento algoritmus nie je prípustný.

Úplnosť:

-DFS taktiež nezaručuje nájdenie vôbec nejakého riešenia. Ak je strom príliš hlboký, môže sa stať, že algoritmus sa zamotá a riešenie sa nikdy nenájde. Taktiež sa môže stať, že aj z konečnej množiny operátorov sa vygeneruje nekonečne hlboký strom. Preto tento algoritmus nie je úplný.

Norbert Matuška
UI Streda 13:00-14:40

Riešenie problému

Tento problém som riešil pomocou rekurzívnej funkcie, ktorá pomocou operátorov prehľadáva 2D pole zaplnené -1. -1 v poli znamená, že toto políčko na šachovnici ešte nebolo navštívené a pokiaľ navštívené už bolo, zapíše sa tam číslo pohybu <0, n*n-1>

Na začiatku si inicializujem všetky potrebné premenné, ktoré budem potrebovať, ako veľkosť šachovnice, samotnú šachovnicu, operátory atď

```
def __init__(self, board_size):
    self.board_size = board_size
    self.board = [[-1 for _ in range(board_size)] for _ in range(board_size)]
    self.moves = [(2, 1), (1, 2), (-1, 2), (-2, 1), (-2, -1), (-1, -2), (1, -
2), (2, -1)]
    self.expanded_nodes = 0
```

Následne mám 3 funkcie, ktoré mi pomáhajú s riešením tohto problému a tie sú:

```
def is_valid_move(self, x, y):
    if 0 <= x < self.board_size and 0 <= y < self.board_size and
self.board[x][y] == -1:
        return True
    return False
```

is_valid_move() jednoducho kontroluje, či sa kôň nachádza na šachovnici a či políčko, na ktoré sa snažíme pohnúť nebolo už navštívené.

```
def solve(self, x_start, y_start):
    self.board[x_start][y_start] = 0
    if self.solve_recursive(x_start, y_start, 1):
        self.print_solution()
        print("Expanded nodes: " + str(self.expanded_nodes))
    else:
        print("No solution was found.")
```

solve() je začiatočná funkcia, kde na šachovnicu zapíšeme nultý krok (kde vlastne začíname) a následne posunieme štartovacie súradnice do rekurzívnej funkcie

```
def solve_recursive(self, x, y, move_num):
    if move_num == self.board_size * self.board_size:
        return True

    for i in range(8):
        next_x = x + self.moves[i][0]
        next_y = y + self.moves[i][1]

        if self.is_valid_move(next_x, next_y):
            self.board[next_x][next_y] = move_num
            self.expanded_nodes += 1
            if self.expanded_nodes >= 10000000:
                return False

            if self.solve_recursive(next_x, next_y, move_num + 1):
                return True
            self.board[next_x][next_y] = -1  # Backtrack

    return False
```

solve_recursive() je hlavná funkcia môjho programu. Na jej začiatku kontrolujeme či už sme prešli celú šachovnicu, ak áno končíme, ak stále nie, ideme ďalej do for loop-u, ktorý sa opakuje 8-krát (počet možných operátorov). Funkcia skúša rôzne smery pohybu na hracej ploche. Ak je pohyb platný, t.j., nevychádzame mimo plochy, tak nastavíme hodnotu na danej pozícii na číslo ťahu (move_num) a zvýšime počet expandovaných uzlov. Pokiaľ dosiahneme 10 000 000 expandovaných uzlov (alebo krokov), je šanca, že sa funkcia nejakým spôsobom zauzlila alebo riešenie nájde po dlhšej dobe, takže ju ukončíme, pretože sme príliš hlboko a nechce sa nám čakať. Funkcia potom rekurzívne volá sama seba na novej pozícii a pokračuje vo vyhľadávaní riešenia. Ak sa podarí nájsť riešenie, funkcia vráti True, inak sa vráti späť (backtrack) a zmení hodnotu na danej pozícii na -1, čím naznačuje, že to nie je správna cesta. Ak sa všetky možnosti preskúmali a riešenie sa nenašlo, funkcia vráti False.

```
if __name__ == "__main__":

    board_size = int(input("Enter board size: "))
    for i in range(5):
        print("\nTest number " + str(i + 1) + ":")

        x = random.randint(0, board_size - 1)
        y = random.randint(0, board_size - 1)

        print("Testing for position: " + str(x), str(y))

        knights_tour = KnightsTour(board_size)

        start_time = time.time()
        knights_tour.solve(x, y)
        end_time = time.time()
        solve_time = end_time - start_time
        print("Execution time is " + str(solve_time) + " seconds")
```

V main-e máme už len nejaké veci na testovanie ako 5 náhodných začiatočných súradníc, alebo meranie času za koľko program nájde riešenie.

Pohyb po šachovnici

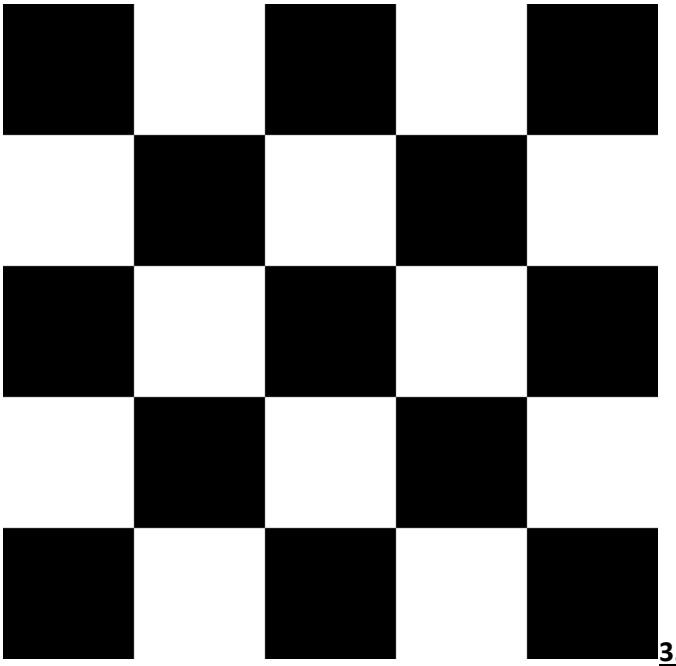
Kôň má 8 možností pohybu a teda 8 operátorov, tie sú nasledovné:

```
self.moves = [(2, 1), (1, 2), (-1, 2), (-2, 1), (-2, -1), (-1, -2), (1, -2),
(2, -1)]
```

Norbert Matuška
UI Streda 13:00-14:40

Neriešiteľné prípady

V mojom prípade, keďže sa zaoberám iba šachovnicami veľkosti 5x5 a 6x6, tak pri šachovnici o veľkosti 5x5 nie vždy existuje riešenie. Takýto záver nastáva práve vtedy, keď kôň začína na bielom políčku šachovnice, podľa obrázku nižšie:



3.

Taktiež to môžeme definovať nasledovne: Ak ľavý horný roh má súradnice (1, 1) tak potom v prípade, že kôň začína na súradniciach, ktoré x-ová je párna a y-ová je nepárna alebo naopak, pre danú začiatočnú súradnicu neexistuje riešenie.

Dobre znázornené aj na nasledujúcej šachovnici:

	1	2	3	4	5
1	304	0	56	0	304
2	0	56	0	56	0
3	56	0	64	0	56
4	0	56	0	56	0
5	304	0	56	0	304

4.

Čo znamená, že ak začíname na rohových súradniciach, napríklad (1, 5) tak môže byť 304 rôznych riešení pre euler-ovho koňa.

Imagine that you are coloring the board like a checkerboard, where the center square is white. Then the board has 13 white squares and 12 black squares.

Every time a knight moves, it changes the color of its square. So if you start on a black square, after 1 move you'll be on a white square (with your original square used up), after 2 moves you'll be on a black square (with 1 black and 1 white square used up), after 3 moves you'll be on a white square (with 2 blacks and 1 white square used up).

Continuing this pattern, after 24 moves, you will have used up 12 black squares, 12 white squares, and your knight will be on a white square.

But the only unused square is white!

It's impossible for you to get to that last square, because it's the same color as the square your knight is on.

5.

Norbert Matuška
UI Streda 13:00-14:40

Testovanie

Program som testoval na piatich náhodných začiatočných pozíciách a následne meral uplynutý čas a expandnuté uzly.

5x5 Šachovnica

Test number 1:

Testing for position: 2 4

```
22  7 12  1 20
11  2 21  6 13
16 23  8 19  0
 3 10 17 14  5
24 15  4  9 18
```

Expanded nodes: 178319

Execution time is 0.49214768409729004 seconds

Test number 2:

Testing for position: 1 3

```
22  3  8 13 24
 9 14 23  0  7
 4 21  2 17 12
15 10 19  6  1
20  5 16 11 18
```

Expanded nodes: 60160

Execution time is 0.1650230884552002 seconds

Test number 3:

Testing for position: 4 2

```
22  3 12  7 20
11  6 21  2 13
16 23  4 19  8
 5 10 17 14  1
24 15  0  9 18
```

Expanded nodes: 36213

Execution time is 0.09763956069946289 seconds

Test number 4:

Norbert Matuška
UI Streda 13:00-14:40

Testing for position: 2 0

```
18 15  4  9 20
 5 10 19 16  3
 0 17 14 21  8
11  6 23  2 13
24  1 12  7 22
```

Expanded nodes: 23448

Execution time is 0.06533455848693848 seconds

Test number 5:

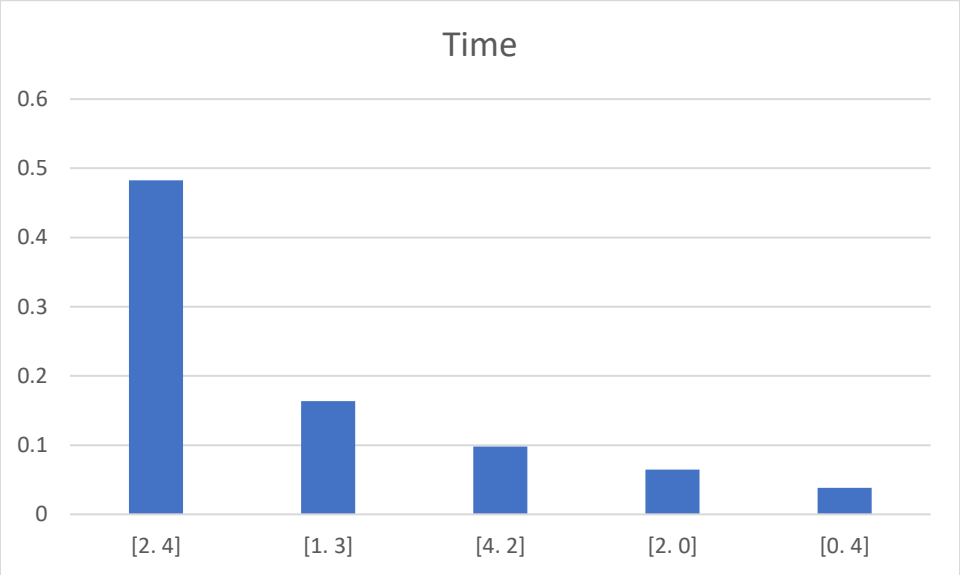
Testing for position: 0 4

```
24 15  4  9  0
 5 10  1 16  3
14 23 12 19  8
11  6 21  2 17
22 13 18  7 20
```

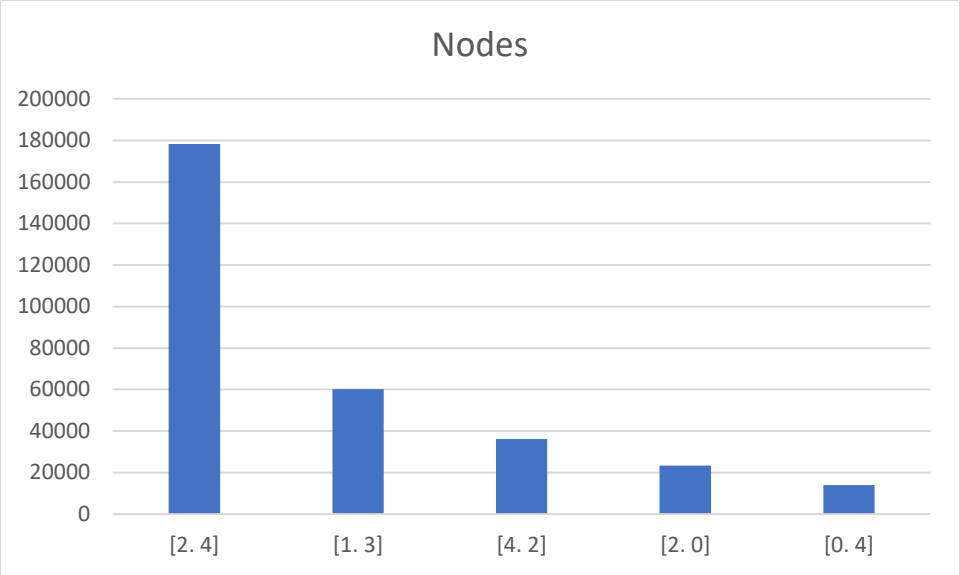
Expanded nodes: 14008

Execution time is 0.03850507736206055 seconds

Porovnanie pre 5x5 šachovnice



Norbert Matuška
UI Streda 13:00-14:40



6x6 šachovnica

Test number 1:

Testing for position: 0 1

```
13  0 19  6 11  8
18 33 12  9 20  5
27 14  1 32  7 10
34 17 28 23  4 21
29 26 15  2 31 24
16 35 30 25 22  3
```

Expanded nodes: 2617588

Execution time is 7.281428813934326 seconds

Test number 2:

Testing for position: 1 3

```
18 29  6  9 16 35
27 10 17  0  5  8
30 19 28  7 34 15
11 26 13 22  1  4
20 31 24  3 14 33
25 12 21 32 23  2
```

Expanded nodes: 897230

Execution time is 3.115830898284912 seconds

Norbert Matuška
UI Streda 13:00-14:40

Test number 3:

Testing for position: 3 4

```
11 20  5 34  9 14
22 33 10 13  4 35
19 12 21  6 15  8
32 23 30 27  0  3
29 18 25  2  7 16
24 31 28 17 26  1
```

Expanded nodes: 221174

Execution time is 0.6284844875335693 seconds

Test number 4:

Testing for position: 1 5

```
30 23  8  1 32 35
25 14 31 34  7  0
22 29 24  9  2 33
15 26 13  4 19  6
12 21 28 17 10  3
27 16 11 20  5 18
```

Expanded nodes: 375293

Execution time is 1.0308973789215088 seconds

Test number 5:

Testing for position: 3 5

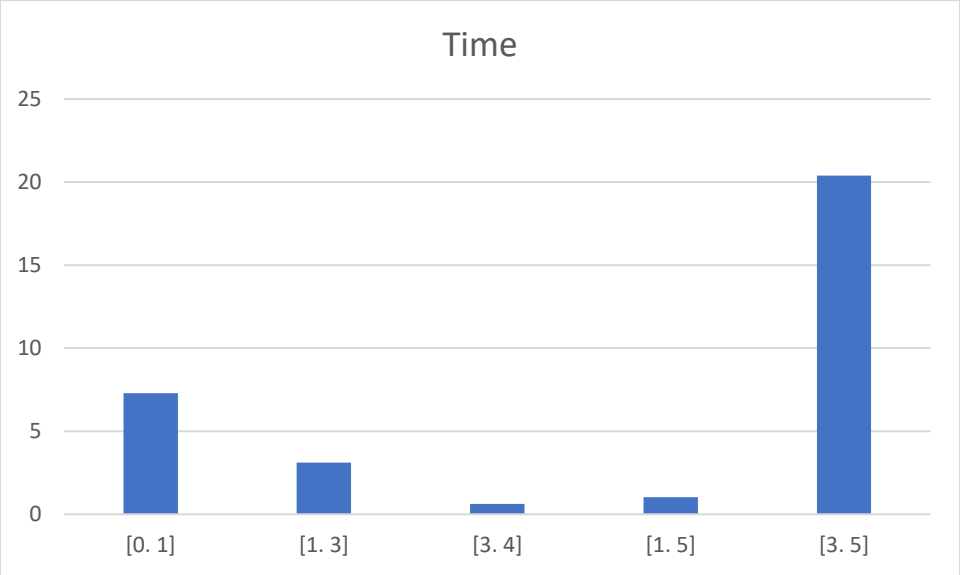
```
 8 17  2 23  6 11
35 24  7 10  1 22
16  9 18  3 12  5
25 34 27 30 21  0
28 15 32 19  4 13
33 26 29 14 31 20
```

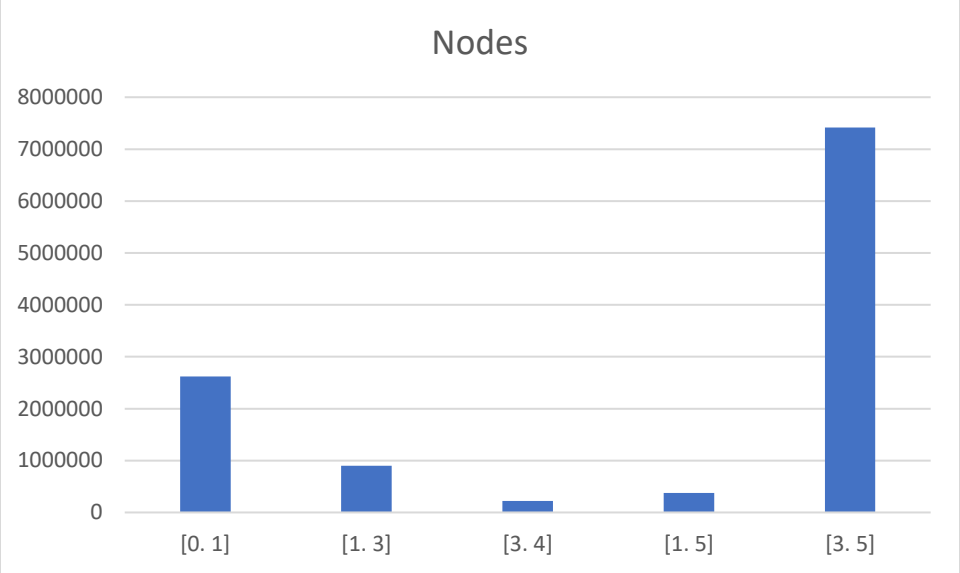
Expanded nodes: 7415226

Execution time is 20.399158477783203 seconds

Norbert Matuška
UI Streda 13:00-14:40

Porovnanie pre 6x6 šachovnice





Norbert Matuška
UI Streda 13:00-14:40

7x7 šachovnica

Skúšal som aj 7x7 šachovnicu ale pri limite 10 miliónov krokov mi to našlo len pre začiatočne súradnice 0 0

Test number 1:

Testing for position: 0 0

```
 0 37 30  7 18 35 14
31 28 19 36 15  6 17
38  1 32 29  8 13 34
27 24 39 20 33 16  5
40 21  2 25 44  9 12
23 26 47 42 11  4 45
48 41 22  3 46 43 10
```

Expanded nodes: 7151178

Execution time is 20.268964767456055 seconds

Norbert Matuška
UI Streda 13:00-14:40

Používateľská príručka

Program v podstate robí všetko za vás, stačí spraviť nasledovné kroky:

1. Otvoriť main.py v nejakom IDE alebo python 3 kompilátore
2. Spustiť program
3. Napísať veľkosť šachovnice
4. Nechať program otestovať 4 náhodné a jednu predurčenú začiatočnú pozíciu

Zdroje obrázkov

1. https://archive.lib.msu.edu/crcmath/math/math/k/k099.htm
2. https://medium.com/basecs/breaking-down-breadth-first-search-cebe696709d9
3. https://commons.wikimedia.org/wiki/File:Checkerboard_pattern.svg
4. https://stackoverflow.com/questions/31411487/knights-tour-on-a-5-x-5-board-start-from-any-square
5. https://stackoverflow.com/questions/31411487/knights-tour-on-a-5-x-5-board-start-from-any-square

Zdroje

1. NÁVRAT, P. et al. Umelá inteligencia. STU Bratislava, 2015. 47 s. ISBN 9788022743440