Problém 3, e) Eulerov kôň

Matúš Makay

STU FIIT

Prehľadávanie stavového priestoru

Pri problematika prehladávania stavového priestoru som dostal za úlohu vyriešiť problém Eulerovho koňa pomocou lačného prehladávania do hĺbky. Problém sa dal riešiť rôznymi spôsobmi:

1. všeobecným algoritnom
2. rekurzia
3. iteračné prehladávanie

Ja som si zvolil prehladanie stavového priestoru pomocou rekurzie nakoľko sme podobný problém riešili minuly semester na DŠA. Používam neinformované hľadanie do hĺbky ktoré nezaručuje že sa nejaký vysledok nájde takže s narastajúcim rozmerom je stále ťažšie dospieť k nejakému výsledku. Časová zložitosť algoritmu je b na d pričom b je faktor vetvenia a d je hĺbka riešenia. Pamäťová zložitosť je rovnaká ako časová.

Modely

Knight

```
class Knight:

    move_x = [2, 1, -1, -2, -2, -1, 1, 2]
    move_y = [1, 2, 2, 1, -1, -2, -2, -1]
    #pocita ako moc hlboko sa program pri rekurzii ponori
    steps = 0

    def get_move(self, num):
        return self.move_x[num], self.move_y[num]
    def update_steps(self):
        self.steps = self.steps + 1
    def get_steps(self):
        return self.steps
```

Šachovnica

Šachovnicu reprezentujem jednoduchým dvojrozmerným polom ktoré má inicializované všetky hodnoty na 0.

Funkcia ktorá vytvára pole je prevzatá z web-u geeks for geeks

```
def create_chessboard(row, column):
    arr=[]
    rows, cols=row, column
    for i in range(rows):
        col = []
        for j in range(cols):
            col.append(0)
        arr.append(col)
    return arr
```

Algoritmus prehladávania

Implementácia

```
def rec_find_path(cur_x, cur_y, jump, chessboard, knight, moves):

    # vsetky policka su oznacene, uspesne skoncenie programu
    if jump == size**2+1:
        return True
    # vypnutie programu ak uz sme v moc velkej hlbke
    if knight.get_steps() >= 10000000:
        print("Solution could not be found")
        sys.exit(1)

    #postupne prejdem vsetky moznosti pohybu
    for num in range(8):
        move_x, move_y = knight.get_move(num)
        new_x = cur_x + move_x
        new_y = cur_y + move_y

        #ak je policko bezpecne oznacim ho
        if utils.is_move_save(new_x, new_y, chessboard, size):

            moves.append((move_x, move_y))
            chessboard[new_x][new_y] = jump
            knight.steps_up()
            #rekurzivne ponorenie s novymi suradnicami
            if (rec_find_path(new_x, new_y, (jump+1), chessboard, knight,
moves)):
                return True
            #ak sa rekurzia pride az sem zmenim policko na nenavstivene
            chess[new_x][new_y] = 0
            #znizim ponorenie o jedno
            knight.steps_down()
            #vyhodim krok zo stack-u
            moves.pop()

    return False
```

Slovný opis

Báza pre problém je podmienka ak už sa počet krokov ktoré kôň skočil rovná 26 pretože počítam skoky od 1. Program takisto zastavím aj ak sa rekurzia ponorí moc hlboko túto funkcionalitu som dorobil v zmysle zadania. Pre párne rozmery šachovnice algoritmus po niekoľkých hodinách nájde riešenie vždy. Algoritmus skákania koňa je veľmi jednoduchý, vyberiem jeden možný pohyb z množiny samostatne pre x a y os. Následne tento pohyb pripočítam po zložkách k aktuálnou pozíciou. Skontrolujem či je nová pozícia bezpečná(nevyskočili sme zo šachovnice alebo či už na nu niekedy kôn neskočil). Ak nie je cyklus sa zopakuje a vyberie sa nová možnosť pohybu. Ak je bezpečná označíme dané políčko v šachovnici aktuálnym skokom a pridáme pohyb do množiny uskutočnených pohybov následne sa ponoríme rekurziou do daľšej vrstvy pre nové súradnice. Ak rekurzia v postupnom ponáraní nájde riešenie a báza sa splní a vrátim hodnotu true ktorá následne ukončí ponáranie a tým aj celý program. Ak rekurzia ale správne riešenie nenájde a v nejakom momente sa už kôň nebude mať kam posunúť, v tomto momente nastúpi backtracking ktorým pri ceste naspäť označím políčko s nevými suradnicami 0 a tým signalizujem že kôň na dané políčko ešte neskočil, takto sa vraciam až kým nenájdem dalšiu možnosť na ponorenie.

Podmienka pre políčko

Implementácia

```
def is_move_save(new_x, new_y, chessboard, size):
    # pozicia musi byt v intervale [0..size-1] a zaroven policko musi byt
nenavstivene
    if new_x > -1 and new_y > -1 and new_x < size and new_y < size and
chessboard[new_x][new_y] == 0:
        return True
    return False
```

Slovný opis

Kontrolujem či políčko je v rozmere šachovnice čož je od nuly po rozmer šachovnice a či na políčko niekedy kôň už neskočil.

Testovanie

Návod na spustenie

```
if len(sys.argv) == 2:
    size = int(sys.argv[1])
    s_x = randint(0, size - 1)
    s_y = randint(0, size - 1)
elif len(sys.argv) == 4:
    size = int(sys.argv[1])
    s_x = int(sys.argv[2])
    s_y = int(sys.argv[3])
elif len(sys.argv) == 3:
    size = randint(5, 6)
    s_x = int(sys.argv[1])
    s_y = int(sys.argv[2])
else:
    size = randint(5, 6)
    s_x = randint(0, size - 1)
    s_y = randint(0, size - 1)
```

Program môžete spustiť nasledovným prikazom, v linuxe sa nastavíte do priecinku zad1/src a potom príkazom „**python knight_tour.py size”** spustíte program so želaným rozmerom šachovnice . Takisto môžete spustiť program príkazom „**python knight_tour.py size s_x s_y“** vtedy sa nastaví rozmer klávesnice a aj to kde má kôň začať alebo „**python knight_tour.py s_x s_y“** kedy sa rozmer nastaví automaticky(náhodne z intervalu [5, 6]) a súradnice kde kôň začne budú podľa vstupu. Najednoduhší spôsob spustenia je pomocou príkazu **„python knight_tour.py“** vtedy sa rozmer aj suradnice kde kôň začne nastavia automaticky. Posledná môžnosť je hodnoty zmeniť priamo v kóde v *else* vetve.

Testovanie som podľa zadania uskutočnil pre rozmer 5x5 a 6x6 pre každý romer som program spustil 5krát. Rozmer je pri testovaní fixný, ten nastavím v kóde. Začiatočnú pozíciu koňa generujem náhodne v súbore rand_gen.py. Pri testovaní môžu nastať tri prípady

1. Nastane situácia keď riešenie existuje potom sa vypíše matica a zoznam skokov ktoré musel kôň vykonať
2. Situácia kedy sa program ponorí až moc hlboko, vtedy podľa zadania zastavím program a vypíšem hlášku „Solution could not be found“
3. Situácia kedy program vyskúša všetky možnosti a nenájde riešenie pre daný rozmer a súradnice, potom vypíšem hlášku „Solution does not exist“

Pre rozmer 5x5

Prvý pokus:

```
~/school/zs_2022/ui/zad/zad1/src ▷ test_ui
Size: 5
Starting position: x: 2, y: 4
Solutuion exists:
[[23  8 13  2 21]
 [12  3 22  7 14]
 [17 24  9 20  1]
 [ 4 11 18 15  6]
 [25 16  5 10 19]]
```

```
Moves which horse has to take:
[(-2, -1), (1, -2), (2, -1), (1, 2), (-1, 2), (-2, -1), (-1,
-2), (2, 1), (2, 1), (-1, -2), (-2, -1), (-1, 2)]
[(2, -1), (1, -2), (-2, -1), (1, 2), (1, 2), (-2, -1), (-2, 1
), (1, -2), (-1, -2), (2, 1), (2, -1)]
```

Druhy pokus:

```
~/school/zs_2022/ui/zad/zad1/src ▷ test_ui
Size: 5
Starting position: x: 4, y: 4
Solutuion exists:
[[25 18  7 12  3]
 [ 8 13  4 17  6]
 [21 24 19  2 11]
 [14  9 22  5 16]
 [23 20 15 10  1]]
```

```
Moves which horse has to take:
[(-2, -1), (-2, 1), (1, -2), (2, 1), (-2, 1), (-1, -2), (1, -
2), (2, 1), (1, 2), (-2, 1), (-2, -1), (1, -2)]
[(1, 2), (-1, 2), (-2, -1), (-1, -2), (2, 1), (2, -1), (-2, -
1), (1, 2), (1, -2), (-2, 1), (-2, -1)]
```

Tretí pokus:

```
~/school/zs_2022/ui/zad/zad1/src ▷ test_ui
Size: 5
Starting position: x: 3, y: 1
Solutuion exists:
[[23 10 13  4 21]
 [12  5 22  9 14]
 [17 24 11 20  3]
 [ 6  1 18 15  8]
 [25 16  7  2 19]]
```

```
Moves which horse has to take:
[(1, 2), (-2, 1), (-2, -1), (1, -2), (2, -1), (1, 2), (-1, 2)
, (-2, -1), (-1, -2), (2, 1), (-1, -2), (-1, 2)]
[(2, -1), (1, -2), (-2, -1), (1, 2), (1, 2), (-2, -1), (-2, 1
), (1, -2), (-1, -2), (2, 1), (2, -1)]
```

Štvrtý pokus:

```
~/school/zs_2022/ui/zad/zad1/src ▷ test_ui
Size: 5
Starting position: x: 4, y: 1
Solution does not exist
```

Piaty pokus:

```
~/school/zs_2022/ui/zad/zad1/src ▷ test_ui
Size: 5
Starting position: x: 1, y: 0
Solution does not exist
```

Pre rozmer 6x6

Pre vysokú mieru výpisu Solution could not be found som podmienku ktorá vypínala program pretože išiel až moc do hĺbky vypol pre účely testovania.

Prvý pokus:

```
~/school/zs_2022/ui/zad/zad1/src ▷ test_ui
Size: 6
Starting position: x: 2, y: 1
Solution could not be found
```

Druhý pokus:

```
~/school/zs_2022/ui/zad/zad1/src ▷ test_ui
Size: 6
Starting position: x: 0, y: 4
Solutuion exists:
[[21 30  9 12  1 36]
 [28 13 22  3  8 11]
 [31 20 29 10 35  2]
 [14 27 16 23  4  7]
 [19 32 25  6 17 34]
 [26 15 18 33 24  5]]
Moves which horse has to take:
[(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), (2, 1), (-2, 1), (1, 2), (1, -2), (-1, -2)]
[(-2, -1), (1, 2), (2, 1), (2, 1), (-1, -2), (1, -2),
(-2, 1), (-2, -1), (1, 2), (-2, -1), (2, -1), (2, 1),
(1, 2), (-1, 2), (-2, -1), (-2, 1)]
```

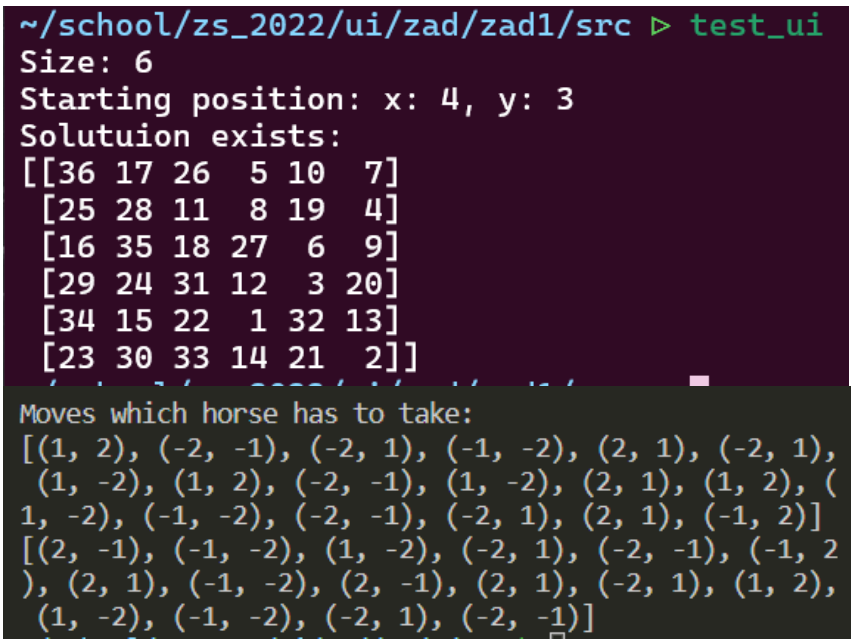
Tretí pokus:

```
~/school/zs_2022/ui/zad/zad1/src ▷ test_ui
Size: 6
Starting position: x: 3, y: 1
Solutuion exists:
[[ 7 30 27 22  5 14]
 [28 21  6 15 26 23]
 [31  8 29 24 13  4]
 [20  1 10 33 16 25]
 [ 9 32 35 18  3 12]
 [36 19  2 11 34 17]]
Moves which horse has to take:
[(2, 1), (-1, 2), (-2, 1), (-2, -1), (1, -2), (-1, -2)
, (2, 1), (2, -1), (-1, 2), (2, 1), (-1, 2), (-2, -1),
 (-2, 1), (1, -2), (2, 1), (2, 1), (-1, -2), (1, -2)]
[(-2, 1), (-1, 2), (1, 2), (1, -2), (1, 2), (-2, -1),
(-1, -2), (1, -2), (1, 2), (-2, -1), (2, -1), (2, 1),
(-1, 2), (2, 1), (-1, -2), (1, -2)]
```

Štvrtý pokus:

```
~/school/zs_2022/ui/zad/zad1/src ▷ test_ui
Size: 6
Starting position: x: 0, y: 4
Solutuion exists:
[[21 30  9 12  1 36]
 [28 13 22  3  8 11]
 [31 20 29 10 35  2]
 [14 27 16 23  4  7]
 [19 32 25  6 17 34]
 [26 15 18 33 24  5]]
Moves which horse has to take:
[(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), (2, 1), (-2, 1), (1, 2), (1, -2), (-1, -2)]
[(-2, -1), (1, 2), (2, 1), (2, 1), (-1, -2), (1, -2),
(-2, 1), (-2, -1), (1, 2), (-2, -1), (2, -1), (2, 1),
(1, 2), (-1, 2), (-2, -1), (-2, 1)]
```

Piaty pokus:



Vykonávanie programu

Čas som vykonania programu som meral pomocou vstavaného prikazu time na linuxe.

Pre rozmer 5x5

1. 0.97
2. 0.3
3. 0.4
4. 6.68
5. 7.04

Čas za ktorý sa program vykonal bol závislý na tom či pre danú pozíciu existuje nejaké riešenie. Ak neexistuje riešenie priemerný čas vykonania bol cca 7s. Za tento čas program prehľadá všetky možnosti.

Pre rozmer 6x6

1. Viac ako 15m
2. 1.76s
3. 319.84s
4. 1.81s
5. 137.10s

S narastajúcim rozmerom sa vykonávanie programu značne predĺži, je to kvôli tomu že stavový priestor je oveľa vačší nakoľko kôň má na každom políčku viacej miest kam sa pohnúť dalej.

Záver k testovaniu

Pri testovaní som narazil na problem dlhého vykonávania sa programu. Algoritmus lačného hľadania do hĺbky sám o sebe pre tento problem nie je vhodný ak chcem aby program skončil behom pár sekúnd pre väčšie rozmery. Algoritmus sa s narastajúcim rozmerom ponorí až moc hlboko a riešenie v danej časti stromu nemusí existovať a vykonávanie programu trvá takto až moc dlho.

Tento problem by sa dal vyriešiť pomoocu použitia heuristiky ktorá funguje na princípe že kôn si z možností skoku vyberie políčko z ktorého môže skočiť na čo najmenej dalších políčok. Taktiež ma napadla možnosť náhodného vyberia pohybu nakoľko v mojom algoritme kroky vyberám postupne pre každý uzol. Túto možnosť som však neodtestoval