Slovenská technická univerzita v Bratislave
Fakulta Informatiky a Informačných Technológií

Umelá inteligencia

Eulerov kôň – algoritmus slepého prehľadávania

Emma Macháčová

Meno cvičiaceho : Ing. Ivan Kapustík
Čas cvičení : štvrtok, 12:00
Zimný semester 2021/2022

Obsah

Riešený problém ... 3
Definovanie problému Eulerov kôň ... 3
Definovanie problému č. 3 ... 3
Opis riešenia ... 4
Reprezentácia údajov ... 4
Použitý algoritmus ... 5
Teoretické vlastnosti ... 6
Časová náročnosť ... 6
Skutočné vlastnosti ... 6
Počet rozvíjaných uzlov ... 6
Trvanie ... 6
Aké riešenie nájde ... 6
Spôsob testovania ... 7
Výsledky: ... 7
Variant A – poradie operátorov (1,2) (2,1) (-1,2) (-2,1) (-1,-2) (-2,-1) (1,-2) (2,-1) ... 7
Variant B – poradie operátorov (-1,2) (1,2) (2,1) (-2,1) (-2,-1) (2,-1) (1,-2) (-1,-2) ... 8
Zhodnotenie riešenia ... 9
Možnosti rozšírenia ... 9
Výhody a nevýhody implementácie ... 9
Optimalizácie ... 9
Porovnanie rôznych veľkostí šachovnice ... 9
Ukážka výpisu ... 10

Riešený problém

Definovanie 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 východiskovom políčku.**

Aktuálny stav najlepšie reprezentuje **šachovnica**. Na začiatku sú všetky jej políčka **vynulované** a postupne sa **zapĺňa číslami, ktoré zodpovedajú príslušnému skoku koňa**. Na reprezentáciu uzla je potrebné pridať pre každý skok aj **poradové číslo operátora** alebo zoznam možných, ešte nevyužitých skokov, aby program pri návrate vedel, **akou alternatívou pokračovať**, prípadne sa vrátiť ešte o ďalší skok naspäť.

Kôň má vo všeobecnosti **8 možností skoku**, ak nie je limitovaný okrajom šachovnice alebo už použitým miestom. To znamená, že existuje 8 operátorov, ktoré je možné označiť napríklad

(1, 2), (1, -2), (2, 1), (2, -1), (-1, 2), (-1, -2), (-2, 1) a (-2, -1),

kde prvé číslo znamená posun od aktuálnej pozície v riadku a druhé v stĺpci.

Algoritmus je **jednoducho klasický algoritmus prehľadávania do hĺbky**, kde sú všetky štruktúry na ukladanie uzlov nahradené (rozšírenou) šachovnicou.

Pre problém Eulerovho koňa treba v oboch úlohách uvažovať s tým, že pre niektoré východiskové 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.**

Definovanie problému č. 3

Algoritmom slepého prehľadávania (do hĺbky) je možné nájsť riešenia iba pri šachovniciach do veľkosti 6x6, max 7x7**. Cieľom je implementovať tento algoritmus pre šachovnice s rozmermi 5x5 a 6x6** a skúsiť nájsť prvých 5 riešení pre každú šachovnicu tak, že pre šachovnicu 5x5 aj 6x6 si vyberieme náhodne 5 východiskových bodov (spolu teda 10 východiskových bodov) s tým, že jeden z týchto bodov je (pre každú šachovnicu) ľavý dolný roh a pre každý z týchto bodov nájdeme prvé riešenie. V prípade, že ho v stanovenom limite nenájdeme, signalizujeme neúspešné hľadanie.

Opis riešenia

Program je implementovaný v jazyku Python (3.9). Skladá sa z dvoch hlavných funkcií (inicializuj a hladajRiesenie), z mainu a z pomocných funkcií:

- **main**,
- **inicializuj** – inicializuje šachovnicu a nastavuje štartovaciu pozíciu, spúšťa hľadanie riešenia,
- **hladajRiesenie** – funkcia obsahujúca hlavný algoritmus,
- **vypis** – vypisuje šachovnicu s riešením (celým alebo čiastočným),
- **isBlack** – kontroluje farbu políčka,
- **updateStat** – aktualizuje štatistiku výsledkov,
- **vypisStat** – vypisuje štatistiku,
- **checkTime** – kontroluje, či nebol prekročený časový limit pri hľadaní riešenia.

Menu pre používateľa ponúka nasledovné možnosti:

```
MENU
    -> pre manualny vstup stlac         (a)
    -> pre automaticke testovanie stlac (b)
    -> pre test vsetkych moznosti stlac (c)

    -> pre upravu casoveho limitu stlac (d)

    -> pre vypis statistiky stlac       (e)

    -> pre ukoncenie programu stlac     (x)
```

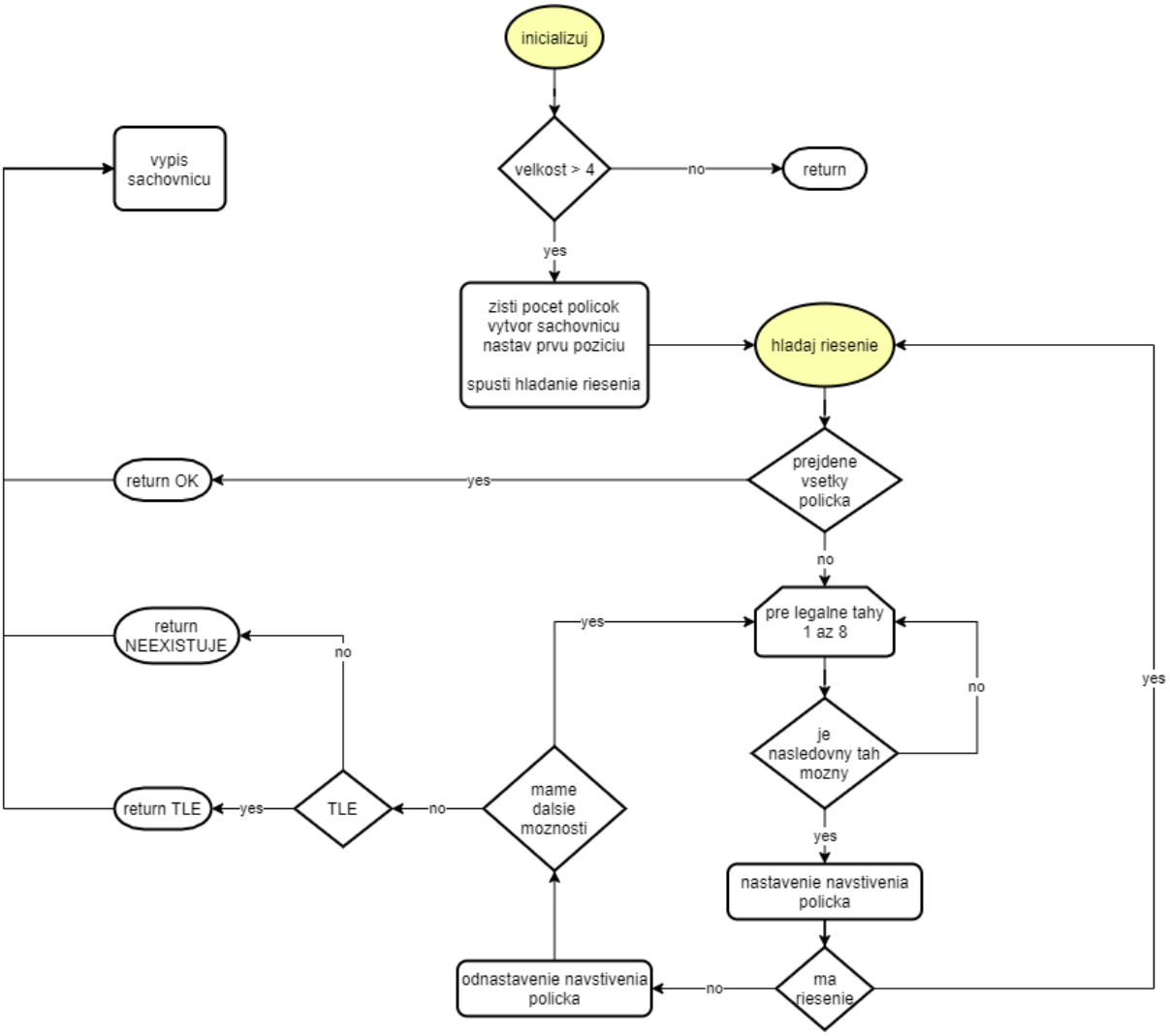
Reprezentácia údajov

Údaje sú reprezentované v **dvojrozmernom poli**, ktoré predstavuje šachovnicu. Nesie údaje o tom, na ktorých políčkach už figúrka stála (a koľký krok to bol). Okrem toho využívam štatistické pomocné premenné (class stat), dočasné a premenlivé **pomocné premenné** (class temp) a nemenné **globálne premenné** (class glob).

```
class temp:  # trieda s globalnymi premennymi
    maxKrok = 0     # pri ciastocnom rieseni - kam sme sa dostali najdalej
    maxSachovnica = [["x" for stlpec in range(0, 20)] for riadok in range(0, 20)]
    cas = 0         # na pocitanie casu
    tl = 0          # flag ci bol prekroceny casovy limit


class stat:...


class glob:  # staticke globalne premenne
    TIMELIMIT = 16
    # pohyb todo vymysliet poradie
    POSUN_X = [1, 2, -1, -2, -1, -2, 1, 2]
    POSUN_Y = [2, 1, 2, 1, -2, -1, -2, -1]
```

Použitý algoritmus

V programe je využitý algoritmus „backtracking-u“ alebo spätného trasovania (lačné hľadanie, hľadanie do hĺbky). Tento algoritmus postupne skúša všetky legálne ťahy, až kým nájde prvé riešenie (alebo zistí, že riešenie neexistuje).



Po inicializácií a nastavení premenných vo funkcií inicializuj sa zavolá rekurzívna funkcia hladajRiesenie, ktorá postupne skúša všetky možnosti, kým nenájde riešenie, kým nepresiahne časový limit, alebo kým nezistí, že riešenie neexistuje.

Funkcia kontroluje v každom volaní to, či nebolo nájdené riešenie, a ak nebolo vyberá postupne z legálnych ťahov, a volá znova samu seba s novou pozíciou figúrky.

Teoretické vlastnosti

Časová náročnosť

Pre každú úlohu máme N^2 políčok a maximum 8 legálnych ťahov – z toho nám vyplýva najhoršia možná časová náročnosť $O(8^{N \wedge 2})$.

Skutočné vlastnosti

Fakt, že máme k dispozícií 8 legálnych ťahov nie je celkom pravdivý v každom prípade. Nie pri každom umiestnení figúrky si môžeme vyberať zo všetkých ťahov, a tým pádom riešenie výrazne závisí od poradia používaných ťahov.

Počet rozvíjaných uzlov

Tento počet závisí od veľkosti šachovnice. Pri binárnom strome veľkosti N je počet uzlov $2^{N+1} - 1$. V našom prípade je však faktor rozvetvovania sa oveľa väčší, keďže jedno políčko má 2 až 8 možných susedov.

Pri šachovnici s veľkosťou 5x5 môžeme vytvoriť strom s výškou 25, čo nám prinesie cca $3 * 10^{14}$ uzlov. Pre šachovnicu 6x6 je výška stromu 36, pre 7x7 je to 49, atď..

Trvanie

Reálnemu trvaniu programu sa venujem v časti Zhodnotenie riešenia.

Aké riešenie nájde

Program nájde prvé riešenie zodpovedajúce poradiu krokov. Keďže cesty nie sú ohodnotené, to ako prejde šachovnicu nevieme hodnotiť. Kvalita riešenia sa dá posudzovať podľa toho, koľko krokov vykoná pred tým, ako nájde riešenie (čo závisí od štartovacieho políčka a poradia dostupných krokov).

Spôsob testovania

Program má rôzne funkcie na testovanie. Používateľ môže vybrať to, v akom režime program pracuje:

a) môže spustiť **jeden test** pre ním zadaný rozmer šachovnice a štartovacie políčko,
b) môže spustiť **automatické testovanie** (tak ako je špecifikované v zadaní), ktoré nájde prvých 5 riešení pre každú šachovnicu (5x5 aj 6x6) tak, že pre šachovnicu 5x5 aj 6x6 si vyberie náhodne 5 východiskových bodov (spolu teda 10 východiskových bodov) s tým, že jeden z týchto bodov je (pre každú šachovnicu) ľavý dolný roh a pre každý z týchto bodov nájdeme prvé riešenie,
c) môže spustiť **test všetkých možností**, ktorý pre zadanú veľkosť šachovnice otestuje postupne všetky políčka a to, či pri štarte z nich vie nájsť riešenie.

Vo všetkých testoch v prípade, že program riešenie v stanovenom limite nenájde, signalizuje neúspešné hľadanie. Pre účely overenia funkčnosti programu som ho testovala využitím možnosti c), a to testu pre všetky možné východiskové políčka (pre veľkosť 5x5 a 6x6), s časovým limitom nastaveným na kratší, aj dlhší interval.

Výsledky:

Výsledky porovnávam podľa toho, v akom poradí program skúša platné ťahy pre figúrku, a tiež hodnotím farbu štartovacieho políčka, rozmer šachovnice a priemerný čas.

Variant A – poradie operátorov (1,2) (2,1) (-1,2) (-2,1) (-1,-2) (-2,-1) (1,-2) (2,-1)

s časovým limitom 16s

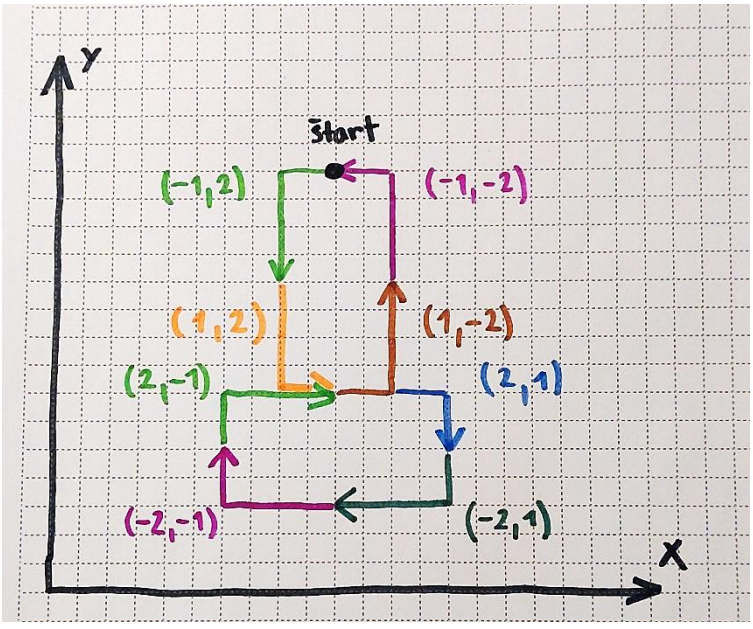
```
Statistika ------------------------------------------------------------
celkovy cas: 430.41 s, casovy limit: 16

celkovy pocet testov:  61    ************************************************************
    z toho uspesnych:  31    *******************************
    a neuspesnych:     30    ******************************
    z toho TLE:        18    ******************
priemerny cas na riesenie:   7.06 s

pocet testov sachovnice velkosti 5x5:   25   *************************
    poc. uspesnych zac. v bielom poli:  0
    poc. uspesnych zac. v ciernom poli: 13   *************
    poc. neuspesnych testov pre rozmer: 12   ************
priemerny cas na uspesne riesenie:      0.22 s

pocet testov sachovnice velkosti 6x6:   36   ************************************
    poc. uspesnych zac. v bielom poli:  8    ********
    poc. uspesnych zac. v ciernom poli: 10   **********
    poc. neuspesnych testov pre rozmer: 18   ******************
                          z toho TLE: 18   ******************
priemerny cas na uspesne riesenie:      2.52 s

pocet testov pre ine velkosti: 0

-----------------------------------------------------------------------
```

V týchto testoch vyšlo ako najlepšie východiskové políčko to, ktoré je úplne v strede šachovnice.

Variant B – poradie operátorov (-1,2) (1,2) (2,1) (-2,1) (-2,-1) (2,-1) (1,-2) (-1,-2)
s časovým limitom 16s

```
Statistika ----------------------------------------------------------
celkovy cas: 257.29 s, casovy limit: 16

celkovy pocet testov:  61    ************************************************************
    z toho uspesnych:  39    ***************************************
    a neuspesnych:     22    **********************
    z toho TLE:        10    **********
priemerny cas na riesenie:   4.22 s

pocet testov sachovnice velkosti 5x5:   25   *************************
    poc. uspesnych zac. v bielom poli:  0
    poc. uspesnych zac. v ciernom poli: 13   *************
    poc. neuspesnych testov pre rozmer: 12   ************
priemerny cas na uspesne riesenie:      0.17 s

pocet testov sachovnice velkosti 6x6:   36   ************************************
    poc. uspesnych zac. v bielom poli:  12   ************
    poc. uspesnych zac. v ciernom poli: 14   **************
    poc. neuspesnych testov pre rozmer: 10   **********
                           z toho TLE: 10    **********
priemerny cas na uspesne riesenie:      1.28 s

pocet testov pre ine velkosti: 0

----------------------------------------------------------------------
```

Táto postupnosť krokov je výrazne efektívnejšia – celkové trvanie testov bolo zmenené na takmer polovicu, znížil sa počet nedokončených testov pre presiahnutie časového limitu, a tiež sa zlepšili časy pre jednotlivé rozmery šachovníc.



(idea za usporiadaním krokov)

Zhodnotenie riešenia

Možnosti rozšírenia

Bez zmeny typu algoritmu, program by bolo možné rozšíriť o to, že by bol používateľ schopný zadefinovať si sám poradie legálnych ťahov.

Výhody a nevýhody implementácie

Výhodou implementácie je to, že máme zaručenú istotu toho, že program riešenie skôr či neskôr nájde (ak existuje). Ďalšou výhodou je to, že ide o pomerne jednoduchý algoritmus.

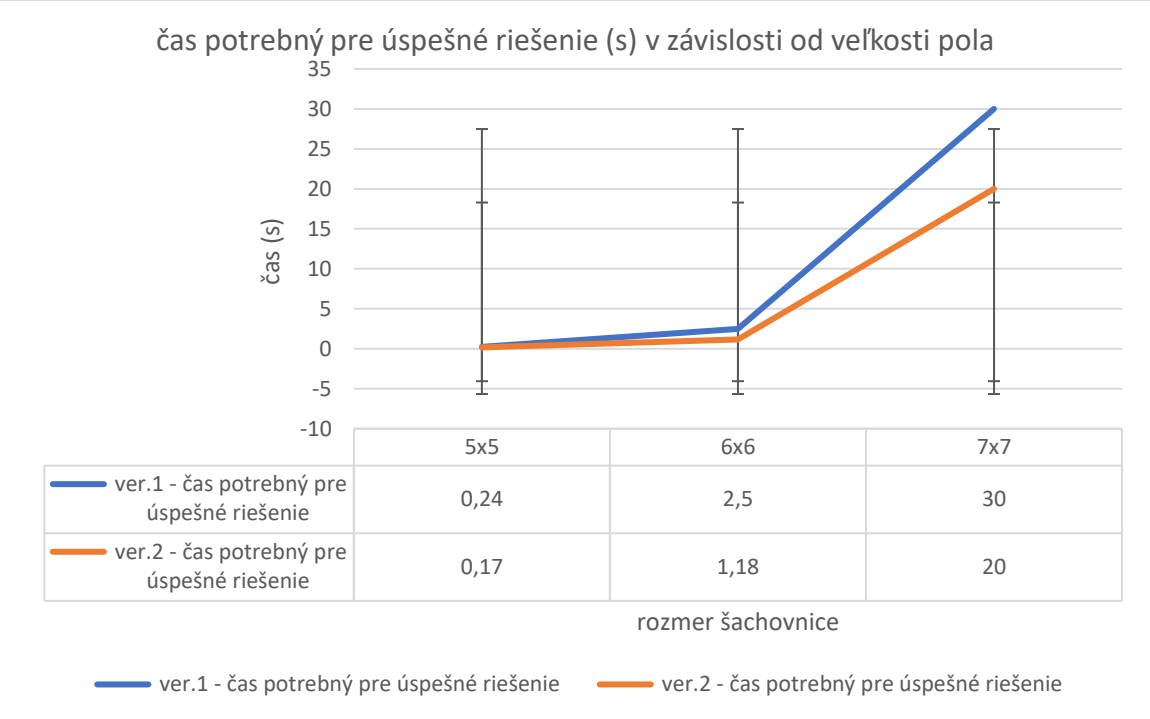
Hlavnou nevýhodou je vysoká časová náročnosť, a nedostatočná flexibilita. Program problém rieši hrubou silou bez toho, aby riešenie prispôsoboval (napríklad podľa aktuálneho umiestnenia figúrky).

Optimalizácie

Jednou z optimalizácií by mohlo byť to, že by program rozdelil šachovnicu na „sektory“, a podľa toho v akom sektore sa figúrka nachádza by volil poradie legálnych ťahov. Ďalšou optimalizáciou by mohlo byť vytvorenie databázy už riešených problémov – v prípade, že by trasu raz našiel, druhý raz by ju nemusel hľadať nanovo, len by ju vypísal z pamäte.

Časovú náročnosť implementácie by vylepšilo tiež to, keby nás nezaujímali čiastočné riešenia (neúplné riešenia, ktoré vznikli buď prekročením časového limitu, alebo tým, že pre dané políčko riešenie neexistuje). Ak by to bolo tak, pre štart z bieleho políčka šachovnice nepárneho rozmeru by program nemusel skúšať hľadať riešenie, a rovno by nás informoval o tom, že neexistuje.

Porovnanie rôznych veľkostí šachovnice



	5x5	6x6	7x7
ver.1 - čas potrebný pre úspešné riešenie	0,24	2,5	30
ver.2 - čas potrebný pre úspešné riešenie	0,17	1,18	20

Ako vidíme, čas potrebný pre riešenie stúpa exponenciálne. Variant dva sa ukázal ako oveľa efektívnejšia – kým pri variante 1 testy pre šachovnice s rozmerom väčším ako 6x6 trvali pridlho, pri variante 2 zbehli aj pri nich niektoré štartovacie pozície relatívne rýchlo. (Presnosť testov väčších rozmerov však kvôli výkonu môjho počítača nie je celkom úplná)

Ukážka výpisu