Slovenská technická univerzita
Fakulta informatiky a informačných technológií

Roman Bitarovský

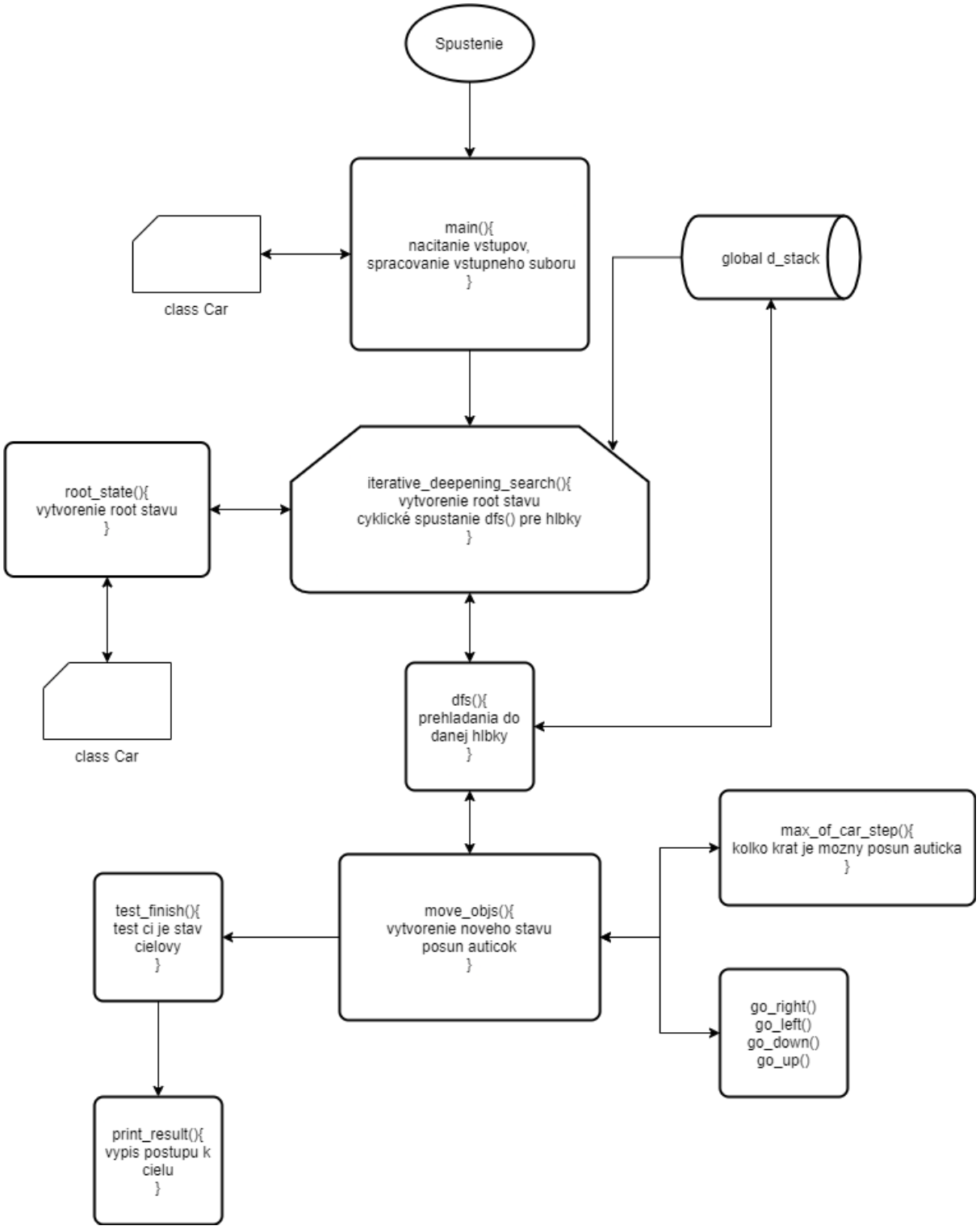
Prehľadávanie stavového priestoru

Bláznivá križovatka

Obsah

Stručný opis riešenia

Program je rozdelený na niekoľko funkcií. Začína funkciou main(), kt. získa vstupy od používateľa a následne po spracovaní zvoleného vstupu zavolá funkciu root_state(), tá vytvorí počiatočný stav obsahujúci mapu podľa vstupných pozícii a následne sa prejde na funkciu iterative_deepening_search(), tá postupne určuje hĺbku do akej sa bude vykonávať prehľadávanie a pre dané hĺbky volá funkciu dfs(), kt. pomocou nerekurzívneho postupu vykonáva prehľadávanie do hĺbky. Pre tento účel využíva zásobník a cyklus.
Z funkcie dfs() je volaná aj funkcia move_objs(). Tá pomocou deepcopy stavu vytvára stav nový v kt. sú následne vykonávané zmeny. Vždy sa zistí či je možné posunúť dané autíčko podľa mapky pre príslušný stav, kt. obsahuje informáciu o tom, kt. políčko je obsadené a kt. je voľné. Autíčka sa posúvajú postupne po jednom políčku pričom s každým posunom vznikne nový stav. Ak je daný stav nový, čo znamená že ešte neexistoval stav s rovnakým rozmiestnením autíčok tak sa zaradí do stacku, ak takýto stav už existoval tak sa poznamená v zozname už navštívených stavov. Pri konci spracovania daného stavu sa vždy kontroluje či náhodu nebol dosiahnutý cieľ (červené autíčko môže vyjsť z križovatky). Kontrola prebieha tak, že sa v danom stave pozrie na mapu, na všetky políčka vpravo od červeného autíčka, či sú voľné a ak áno znamená to, že červené autíčko môže vyjsť z križovatky a teda je to cieľový stav s tým, že ešte treba vypočítať posledný krok a to posun autíčka o n políčok doprava tak aby bolo na konci križovatky.


Spustenie
main(){
nacitanie vstupov,
spracovanie vstupneho suboru
}
class Car
global d_stack
root_state(){
vytvorenie root stavu
}
iterative_deepening_search(){
vytvorenie root stavu
cyklické spustanie dfs() pre hlbky
}
class Car
dfs(){
prehladania do
danej hlbky
}
max_of_car_step(){
kolko krat je mozny posun auticka
}
test_finish(){
test ci je stav
cielovy
}
move_objs(){
vytvorenie noveho stavu
posun auticok
}
go_right()
go_left()
go_down()
go_up()
print_result(){
vypis postupu k
cielu
}

Reprezentácia údajov

Program pri svojej činnosti využíva na reprezentáciu údajov dve hlavné triedy z kt. Sú tvorené inštancie.
Prvá trieda Car slúži na reprezentovanie autíčka a teda každý stav obsahuje týchto autíčok viac. Trieda Car obsahuje informácie ako id, čo je ekvivalent farby autíčk, veľkosť čo je vždy buď 2 alebo 3, polohu x a y a poslednou informáciou je orientácia, kt. Je buď vertikálna (ver) alebo horizontálna (hor).

```
22  class Car:
23      # constructor
24      def __init__(self, id, size, x, y, orientation):
25          # colour
26          self.id = id
27          # ako je dlhe, min je dva aby malo orientáciu
28          self.size = size
29          self.x = int(x)
30          self.y = int(y)
31          # ver (vertikalne) -> |
32          # hor (horizontalne) -> -
33          self.orientation = orientation
```

Ďalšia vlastná štruktúra je trieda State. Je to reprezentácia stavu resp. uzla. Obsahuje reprezentáciu mapky ako 2d pola, všetky autíčka prezentované štruktúrou pomocou triedy Car, hĺbku aby sa vedelo, či je možné daný stav ďalej rozvíjať do hĺbky.

```
40  class State:
41      def __init__(self, crossroad, cars, depth, parent):
42          self.crossroad = crossroad
43          self.cars = cars
44          self.depth = depth
45          self.note = ""
46          self.parent = parent
```

Algoritmus

Na riešenie problematiky zadania je použitý algoritmus cyklicky sa prehlbujúceho hľadania. Algoritmus poskytuje akoby určitú kombináciu hľadania do šírky a do hĺbky.

V podstate sa jedná o vykonávanie prehľadávania do hĺbky s tým, že vieme určiť do akej hĺbky chceme prehľadávať a potom sa prehľadávanie jednoducho skončí.

Moja implementácia algoritmu využíva pre svoje fungovanie základný cyklus pre postupné prechádzanie hĺbok a následne cyklus a zásobník pre prehľadávanie stavov plus ešte pomocný zoznam pre ukladanie už navštívených stavov (dala by sa použiť aj rekurzia)

Hlavný cyklus na postupné skúmanie hĺbok.

```
290     d = 0
291
292     # vypis pociatocneho stavu (farebna mapka)
293     term_print(cars)
294     # postupne zvysovanie hlbky
295     while d != max_depht:
296         print(f"***** Pokus c. {d} *****")
297         state = root_state(max_depht, cars)
298         d_stack.append(state)
299         if dfs(state, d):
300             flag = True
301             break
302         d += 1
```

Prehľadávanie do hĺbky

```
247     while 1 != 0:
248
249         if len(d_stack) == 0:
250             return False
251         state = d_stack.pop()
252         visited.append(state)
253         if state.depth > depth:
254             continue
255
256         # riešenie daného stavu
257         for car in state.cars:
```

Manažovanie čo sa ďalej s prehľadávaným stavom udeje

```
        # ak uz stav bol navstiveny
        if temp_state in visited:
            i = visited.index(temp_state)
            # ak je novy stav navstiveny ale je lepsi (ma mensiu hlbku)
            if temp_state == visited[i] and temp_state.depth < visited[i].depth:
                # tak sa stavy vymenia
                d_stack.append(temp_state)
                visited.remove(visited[i])
                visited.append(temp_state)
                break
        # ak nebol navstiveny zaradi sa do zasobniku na spracovanie
        else:
            d_stack.append(temp_state)
```

Počet generovaných stavou pre vzorové zadanie:

```
***** Pokus c. 8 *****

auticko(grey 7) go_left o 3
auticko(white 1) go_right o 1
auticko(yellow 2) go_up o 1
auticko(magenta 3) go_up o 1
auticko(cyan 5) go_left o 2
auticko(green 6) go_down o 2
auticko(blue 8) go_down o 3
 Last: auticko(red 4) go_right o 3
2 1 1 * * *
2 * * * * *
2 4 4 * * *
3 * * 6 * 8
3 7 7 6 * 8
5 5 5 6 * 8
----------------------
sum_of_state:  361483
Cas: 322.42s
```

Testovanie

Všetky podstatné informácie testu je možne vždy vidieť na príslušnom výpise.
Na začiatku výpisu je vykreslená mapka, s počiatočnými stavmi autíčok. Následne je výpisok programu ohľadom pokusov, v podstate jeden pokus akoby prezentuje jednu hĺbku v ktorej bol cieľ hľadaný. Samozrejme hĺbka nula je počiatočný stav.

Ak program našiel riešenie tak je vypísaná najlepšia postupnosť krokov, kt. program našiel pre dosiahnutie cieľa.
Následne je zase farebne vykreslená mapka obsahujúca cieľový stav predtým ako červené autíčko opustí križovatku. Teda stav bez kroku označeného ako Last, krok Last však vždy môže byť iba posun červeného autíčka o n políčok smerom von z križovatky.

```
Vyberte vstup: 3
Zadajte hĺku do akej chcete vyhladavat: 10
* * * 3 * *
* * * 3 * *
2 4 4 3 * *
2 * 1 1 * *
* * * * * *
* * * * * *
----------------------
***** Pokus c. 0 *****
***** Pokus c. 1 *****
***** Pokus c. 2 *****
***** Pokus c. 3 *****

auticko(white 1) go_left o 1
auticko(magenta 3) go_down o 3
 Last: auticko(red 4) go_right o 3
* * * * * *
* * * * * *
2 4 4 * * *
2 1 1 3 * *
* * * 3 * *
* * * 3 * *
----------------------
Cas: 0.23s
```

```
Vyberte vstup: 1
Zadajte hĺku do akej chcete vyhladavat: 12
1 1 * * * *
2 * * 6 * *
2 4 4 6 * *
2 * * 6 * *
3 * * * * *
3 * 5 5 5 *
----------------------
***** Pokus c. 0 *****
***** Pokus c. 1 *****
***** Pokus c. 2 *****
***** Pokus c. 3 *****
***** Pokus c. 4 *****
***** Pokus c. 5 *****
***** Pokus c. 6 *****

auticko(white 1) go_right o 4
auticko(yellow 2) go_up o 1
auticko(magenta 3) go_up o 1
auticko(cyan 5) go_left o 2
auticko(green 6) go_down o 2
 Last: auticko(red 4) go_right o 3
2 * * * 1 1
2 * * * * *
2 4 4 * * *
3 * * 6 * *
3 * * 6 * *
5 5 5 6 * *
----------------------
Cas: 6.00s
```

```
Vyberte vstup: 4
Zadajte hĺku do akej chcete vyhladavat: 5
* * * * * *
* 3 1 1 * *
2 3 * 5 * *
2 3 * 5 * *
4 4 * 5 * *
* * * * * *
----------------------
***** Pokus c. 0 *****
***** Pokus c. 1 *****
***** Pokus c. 2 *****
***** Pokus c. 3 *****

auticko(white 1) go_right o 2
auticko(cyan 5) go_up o 1
 Last: auticko(red 4) go_right o 4
* * * * * *
* 3 * 5 1 1
2 3 * 5 * *
2 3 * 5 * *
4 4 * * * *
* * * * * *
----------------------
Cas: 0.34s
```

Testovanie prebiehalo spôsobom, že som si vymyslel viacero vstupných scenárov a nechal ich zbehnúť v programe a riešenie, nájdené programom následne „ručne“ skontroloval. Konštatujem, že program pracuje korektne a vo väčšine prípadov nájde naozaj najefektívnejšie riešenie. Samozrejme sú aj výnimočné prípady kedy program nájde o jeden krok naviac. Zväčša nepotrebná krok tam a neskôr späť.

Vybrané testovacie výstupy sem uvádzam v ich originálnom znení (ako ich dal program). Samozrejme prikladám hlavne výstup pre vstup zo zadania.

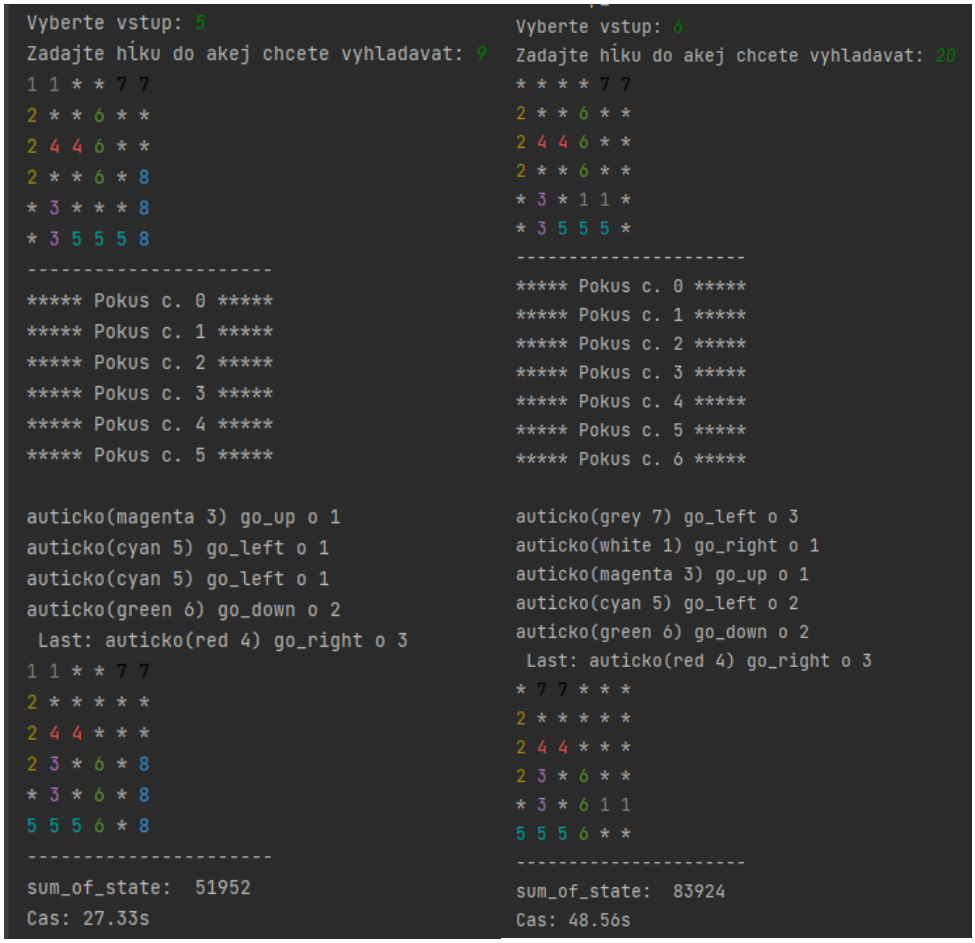
Vstup zo zadania

```
Vyberte vstup: 2
Zadajte hĺku do akej chcete vyhladavat: 10
1 1 * * * 8
2 * * 6 * 8
2 4 4 6 * 8
2 * * 6 * *
3 * * * 7 7
3 * 5 5 5 *
----------------------
***** Pokus c. 0 *****
***** Pokus c. 1 *****
***** Pokus c. 2 *****
***** Pokus c. 3 *****
***** Pokus c. 4 *****
***** Pokus c. 5 *****
***** Pokus c. 6 *****
***** Pokus c. 7 *****
***** Pokus c. 8 *****

auticko(grey 7) go_left o 3
auticko(white 1) go_right o 1
auticko(yellow 2) go_up o 1
auticko(magenta 3) go_up o 1
auticko(cyan 5) go_left o 2
auticko(green 6) go_down o 2
auticko(blue 8) go_down o 3
 Last: auticko(red 4) go_right o 3
2 1 1 * * *
2 * * * * *
2 4 4 * * *
3 * * 6 * 8
3 7 7 6 * 8
5 5 5 6 * 8
----------------------
Cas: 321.51s

Process finished with exit code 0
```

Ďalšie vstupy



```
Vyberte vstup: 5
Zadajte hĺku do akej chcete vyhladavat: 9
1 1 * * 7 7
2 * * 6 * *
2 4 4 6 * *
2 * * 6 * 8
* 3 * * * 8
* 3 5 5 5 8
----------------------
***** Pokus c. 0 *****
***** Pokus c. 1 *****
***** Pokus c. 2 *****
***** Pokus c. 3 *****
***** Pokus c. 4 *****
***** Pokus c. 5 *****

auticko(magenta 3) go_up o 1
auticko(cyan 5) go_left o 1
auticko(cyan 5) go_left o 1
auticko(green 6) go_down o 2
 Last: auticko(red 4) go_right o 3
1 1 * * 7 7
2 * * * * *
2 4 4 * * *
2 3 * 6 * 8
* 3 * 6 * 8
5 5 5 6 * 8
----------------------
sum_of_state:  51952
Cas: 27.33s
```

```
Vyberte vstup: 6
Zadajte hĺku do akej chcete vyhladavat: 20
* * * * 7 7
2 * * 6 * *
2 4 4 6 * *
2 * * 6 * *
* 3 * 1 1 *
* 3 5 5 5 *
----------------------
***** Pokus c. 0 *****
***** Pokus c. 1 *****
***** Pokus c. 2 *****
***** Pokus c. 3 *****
***** Pokus c. 4 *****
***** Pokus c. 5 *****
***** Pokus c. 6 *****

auticko(grey 7) go_left o 3
auticko(white 1) go_right o 1
auticko(magenta 3) go_up o 1
auticko(cyan 5) go_left o 2
auticko(green 6) go_down o 2
 Last: auticko(red 4) go_right o 3
* 7 7 * * *
2 * * * * *
2 4 4 * * *
2 3 * 6 * *
* 3 * 6 1 1
5 5 5 6 * *
----------------------
sum_of_state:  83924
Cas: 48.56s
```

Zhodnotenie riešenia

Konštatujem, že program pracuje správne nakoľko autíčka sa pohybujú korektne, nevychádzajú z mapky, nenarážajú do seba či nepreskakujú cez seba.
Pri menších vstupoch je poskytnutie výstupu takmer okamžité pri väčších vstupoch môže trvať istú (krátku) dobu kým program nájde riešenie nakoľko je potrebné preskúmať naozaj veľké množstvo možných stavov.
Určitou nevýhodou mojej implementácie je použitie jazyka python 3, nakoľko ako všeobecne vieme python je pomalý vzhľadom nato, že sa jedná o jazyk vysokej úrovne. Nedostupnosť určiť presný dátový typ a jednoduchší formát loopov tohto jazyka spôsobuje nie zrovna najefektívnejšie nakladanie s pamäťou. Komplexnejšie dátové štruktúry ako napríklad list tak isto uberajú z rýchlosti programu ak by sme program implementovali napríklad v jazyku C môžeme očakávať rýchlejšie spracovanie.
Optimalizovať by sa však dal aj môj program. Ak by sme implementovali napríklad vyhodnocovanie ako je, kt. stav dobrý pre ďalšie rozvíjanie je možné, že by sme vedeli najlepšie riešenie nájsť skôr
Rovnako vyhodnocovanie či bol už daný stav navštívený by sa dalo z aktuálneho porovnávania objektov tiež prerobiť napríklad na porovnávanie hashov určitých náležitostí.

Zadania je spracované v jazyku python 3.9.5 v prostredí PyCharm a je spustiteľné ak máme funkčné všetky balíčky uvedené na začiatku kódu.