Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Tibor Vanek

Zadanie 3a – **Zenová záhrada**

Tabu Search

Predmet: Umelá Inteligencia
Čas cvičenia: Štvrtok 14:00
Cvičiaci: Ing. Ivan Kapustík

Špecifikácia zadania

Pri tomto zadaní som sa sústredil na záhradu zo zadania. Zenová záhrada je teda plocha reprezentovaná maticou, ktorá je rozmeru **12x10** a obsahuje 6 prekážok (kameňov). Voľné miesta sú označené **0** a kamene sú označené **-1**.
Záhrada vyzerá v nepohrabanom tvare takto:

```
[ 0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0 -1  0  0  0  0  0  0]
[ 0 -1  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0 -1  0  0  0  0  0  0  0]
[ 0  0 -1  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0 -1 -1  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0]
```

Mních sa snaží záhradu pohrabať tak, že začne na niektorom krajnom políčku a pohybuje sa **smerom od steny**, pričom hrabe rovný pás cez záhradu, dokiaľ nevyjde z mapy, alebo nenarazí na prekážku. Ak mních pohrabe riadok a vyjde zo záhrady, **posunie sa** na nové krajné políčko, nastaví si smer a hrabe rovnakým spôsobom. Po použití krajného políčka na ňom mních už **nesmie** znova začať, preto sa odstráni z listu voľných.
Mních nesmie vstúpiť na miesto, kde už hrabal a v prípade že na také narazí, tiež sa považuje za prekážku. Ak narazí na prekážku a nemá sa kam otočiť, jeho hrabanie končí a prechádza sa na vytváranie nasledovníkov.
Úspešné hrabanie je také, kde mních pohrabe celú záhradu. Hrabanie sa reprezentuje číslami tak, že sa vpisuje do matice číslo tam, kde hrabal v toto určité hrabanie. Priebežný stav hrabania by vyzeral napríklad takto:

0	0	1	0	0	0	0	0	10	10	8	9
0	0	1	0	0	K	0	0	10	10	8	9
0	K	1	0	0	0	0	0	10	10	8	9
0	0	1	1	K	0	0	0	10	10	8	9
0	0	K	1	0	0	0	0	10	10	8	9
2	2	2	1	0	0	0	0	10	10	8	9
3	3	2	1	0	0	0	0	K	K	8	8
4	3	2	1	0	0	0	0	5	5	5	5
4	3	2	1	0	0	0	11	5	6	6	6
4	3	2	1	0	0	0	11	5	6	7	7

Zdroj: http://www2.fiit.stuba.sk/~kapustik/zen.html

Keď mních pohrabe záhradu, prechádza sa do štádia vytvárania nasledovníkov. Z prvého mnícha sa vytvorí list nasledovníkov a všetci pohrabú záhradu aby sa zistilo ich fitness. Po každej generácii mníchov sa určí mních s najväčším fitness a z neho sa vytvorí ďalšia generácia. Pomocou tabu search sa takto vyberie najlepší mních z N generácií. Vyberá sa až dokiaľ sa neprejde stanovený počet generácií, alebo sa nenájde mních s najlepším možným fitness.

Príklad úspešného riešenia (114 fitness):

```
[11  6  8  8  8  8  1  9 14 14  5  4]
[11  6  6  6  6 -1  1  9 14 14  5  4]
[11 -1  6  6  6  6  1  9 14 14  5  4]
[ 6  6  6  6 -1  6  1  9 14 14  5  4]
[15 15 -1  6  6  6  1  9 14 14  5  4]
[ 2  2  2  2  2  2  1  9 14 14  5  4]
[ 7  7  7  3  3  2  1  9 -1 -1  5  4]
[13 13  7  3  3  2  1  9 10 10  5  4]
[13 13  7  3  3  2  1  9 10 10  5  4]
[12 12  7  3  3  2  1  9 10 10  5  4]
```

Opis riešenia

Zadanie som vypracoval v programe *Python 3.10* a použil som IDE *PyCharm*.

V mojom programe je mních reprezentovaný štruktúrou, ktorá má 3 vlastnosti:

- `fitness` – počet políčok, ktoré mních dokáže pohrabať
- `directions` – list smerov, kam sa môže mních vyhnúť
- `entering_tiles` – list krajných políčok, na ktorých začína hrabať

```
class Monk:
    def __init__(self, directions, entering_tiles):
        self.fitness = 0
        self.directions = directions
        self.entering_tiles = entering_tiles
```

Smery vyhýbania si ukladám ako *tuple*:
(-1, 0) – hore
(0, -1) – vľavo
(1, 0) – dole
(0, 1) – vpravo

Krajné políčka ukladám taktiež ako *tuple*, ale ukladám sem na začiatku **všetky krajné pozície** – napr. (0,0) je ľavý horný roh, (0,11) je pravý horný roh, atď...
Pre prvého mnícha sa tieto hodnoty **pseudonáhodne zamiešajú** pomocou `shuffle()` z Python knižnice `random` a vložia sa do inštancie objektu `first_monk`.

```
def createMonk():
    directions = [up, left, down, right]
    shuffle(directions)

    entering_tiles = load_entering_tiles()
    shuffle(entering_tiles)

    first_monk = Monk(directions, entering_tiles)
    return first_monk
```

Prvý mních s týmito hodnotami začne hrabať záhradu. Objaví sa na prvom krajnom políčku v liste – monk.entering_tiles[0], skontroluje sa, či je miesto rohové,

- ak nie je, smer mnícha je **oproti kraju**, t. j. ak je na [0][2], ide smerom do prava

```
# ak mnich nie je v rohu, ide smerom od steny
if tile_x == 0 and not is_in_corner((tile_x, tile_y)):
    direction = down
if tile_y == 0 and not is_in_corner((tile_x, tile_y)):
    direction = right
if tile_x == 9 and not is_in_corner((tile_x, tile_y)):
    direction = up
if tile_y == 11 and not is_in_corner((tile_x, tile_y)):
    direction = left
```

- ak je miesto rohové, smer mnícha je ten, ktorý **sa nachádza** v monk.**directions** ako **prvý**

Ak mních narazí a chce sa vyhnúť, len sa zmení smer tak, že sa odkontrolujú smery, kam mních môže pokračovať a jeden sa vyberie. Keď mních skončí hrabanie úspešne u okraju, označí sa miesto, aby z neho už nemohol začať. Taktiež ak prejde pri hrabaní cez okrajové miesto.

Po dokončení hrabania prvého mnícha sa z neho vygenerujú nasledovníci. Generujem ich tak, že zoberiem mnícha a vymieňam hodnoty jeho génu na určenie vstupných políčok (*entering_tiles*). Vymieňam ich postupne štýlom prvá hodnota s druhou, druhá s treťou, tretia so štvrtou postupne až posledná s prvou. Každá výmena znamená 1 nový nasledovník. Takto sa ich vygeneruje **40**. Následne ešte vygenerujem **ďalších štyroch** tak, že rovnakým štýlom povymieňam smery (*directions)*. Dokopy teda bude **v jednej generácii 44 nasledovníkov**.
Po vytvorení nasledovníkov **každý pohrabe tú istú záhradu**, aby sa určilo ich fitness a po každom takomto hrabaní sa záhrada premaže (garden.clear()). Tento zoznam nasledovníkov sa z funkcie vráti ako list (list_of_monks).

Na hľadanie najlepšieho mnícha používam podľa zadania **tabu search**.

```
1 sBest ← s0
2 bestCandidate ← s0
3 tabuList ← []
4 tabuList.push(s0)
5 while (not stoppingCondition())
6     sNeighborhood ← getNeighbors(bestCandidate)
7     bestCandidate ← sNeighborhood[0]
8     for (sCandidate in sNeighborhood)
9         if ( (not tabuList.contains(sCandidate)) and (fitness(sCandidate) > fitness(bestCandidate)) )
10            bestCandidate ← sCandidate
11        end
12    end
13    if (fitness(bestCandidate) > fitness(sBest))
14        sBest ← bestCandidate
15    end
16    tabuList.push(bestCandidate)
17    if (tabuList.size > maxTabuSize)
18        tabuList.removeFirst()
19    end
20 end
21 return sBest
```

Inšpiroval som sa hlavne pseudokódom z prednášky 5, pričom som pridal/zmenil niektoré časti, aby to fungovalo na môj algoritmus.

Tabu search je vylepšenie tzv. „Hillclimbing“ algoritmu.

Funguje klasicky tak, že si uložím fitness aktuálneho mnícha najlepšiu hodnotu a vytvorím z neho nasledovníkov. Prechádzam fitness nasledovníkov a ak sa tu nachádza mních s väčšou fitness ako aktuálne najlepší mních, opakuje sa proces sa z tohto mnícha. Ak sa nenájde v generácii lepší fitness, vyberie sa mních s horším fitness, uloží mnícha do tabu listu a opakuje sa proces z vybraného.
Počas porovnávania si tabu search pamätá najlepšieho mnícha, takže aj keď sa nenájde v najnovšej generácii, stále ho má uloženého.

V mojom riešení vytváram nasledovníkov vymieňaním hodnôt génu vstupných políčok mnícha a preto vzniká väčšinou generácia v takomto tvare:

```
vygenerovanych 44 nasledovnikov
95 76 76 76 76 76 79 76 76 76 84 76 76 76 76 76 76 76 76 76 76 76 76 76 76 76 76 76 76 76 76 76 76 76 76 76 76 76 77 76 76 75 65
```

To, že v tejto generácii väčšina mníchov má rovnaké fitness spôsobuje **problém pri vkladaní hodnôt do tabu listu**, keďže aj keby som pridal jednu hodnotu do tabu listu, stále by sa mohli použiť ostatné a to by spôsobilo **zacyklenie**. Toto riešim tak, že pri porovnávaní fitness hodnôt mníchov porovnávam aj, či sa v tabu liste nachádza mních **s rovnakou fitness**, ako aktuálne porovnávaný mnícha. V tomto prípade by sa neporovnávala hodnota tohto mnícha a prešlo by sa na ďalšieho. Toto ale má za následky, že sa niekedy môže stať, že všetci mnísi z generácie už majú svoju fitness v tabu liste, takže by sa nevybral najlepší z generácie. V tom prípade určím jedného z generácie manuálne z posledných štyroch mníchov (títo majú vymenené smery, zvyčajne sú teda iní od zbytku).

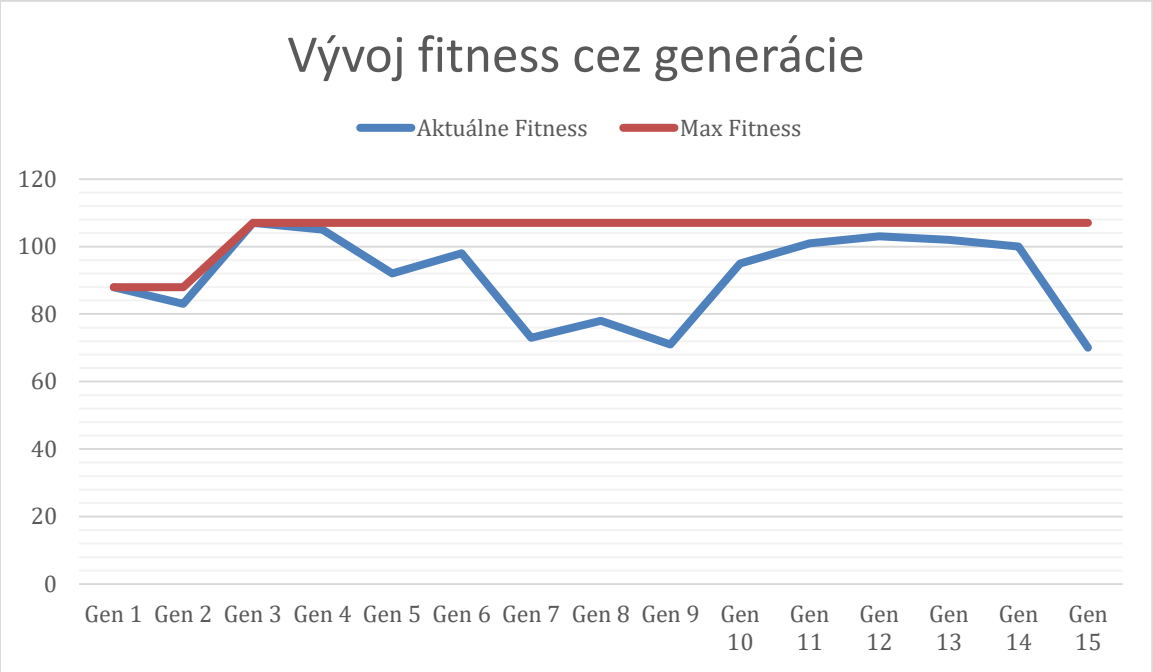
Zhodnotenie a testovanie

Program beží len na vzorovej záhrade, takže výsledky sú z tohto hľadiska porovnateľné.

Max generácií	TabuList Size	Počet testovaní	Priemerné maximum za všetky testy	Najväčšie max fitness za všetky testy
15	30	10	100,4	107
50	30	10	106,3	113
100	15	10	106,9	111
30	50	10	94	105

Z testovania som zistil, že ak veľkosť tabu listu presiahne max počet generácií, priemerné nájdené fitness je zvyčajne horšie. Od dĺžky tabu listu závisí aj efektivita, keďže príliš krátky by znamenalo možnosť zacyklenia a pri príliš dlhom tabu liste by sa zväčšoval čas vykonania. Zvyčajne ešte platí, že čím väčší počet generácií, tým väčšie dosiahnuté maximum.

Graf: Vývoj fitness cez generácie pri
`veľkosť tabu listu = 20`
`počet generácií = 15`



Celkovo si myslím, že efektivita môjho riešenia tohto zadania nie je najlepšia z dôvodu relatívne komplikovaného algoritmu. Vylepšiť by sa dalo napríklad spôsob vyberania nasledovníkov – efektívnejšie vymieňanie, aby nevznikali podobní mnísi v generácii.

Na prípadné vyskúšanie programu s inými vlastnosťami stačí vo funkcii `tabu_search()` zmeniť parametre `maxTabuSize = X` a `while nr_generations < Y...`
Pre zobrazenie, ako pohrabal najlepší mních záhradu stačí odkomentovať tento kód v main:

```
349     #best_monk.fitness = 0
350     #bestgarden = createGarden()
351     #bestgarden = rake_garden(bestgarden, best_monk)
352     #print("Zahrada najlepsieho mnicha:")
353     #print_garden(bestgarden)
354
355
356 main()
357
```