Zenová záhrada

Fakulta informatiky a informačných technológií v Bratislave
Matúš Staš

Riešený problém

Na začiatku máme nepohrabanú zenovú záhradu s voliteľne rozmiestnenými kamenmi alebo listami. Našou úlohou je celú záhradu pohrabať. Toto má na starosti mních, ktorý sa musí vždy pohybovať vo vodorovnom alebo zvislom smere. Ak narazí na prekážku, musí zmeniť smer (vľavo alebo vpravo) ak má kde. Mních sa musí pri každom hrabaní dostať na okraj záhrady. Ak sa tak nestane, úlohu nesplnil.

Implementácia

Projekt je riešený v jazyku Python.

Opis riešenia

Tento problém som riešil evolučným algoritmom, ktorý sa člení na pár častí.

1. **Vytvorenie prvej generácie**
   Náhodne nastavíme gény, každému chromozómu. Avšak dávame si pozor na to, aby sa jednotlivé gény (čísla) neopakovali. Tieto vytvorené chromozómy si udržujeme v zozname ako populácia.

2. **Získanie fitness**
   Pustíme mnícha hrabať záhradu. Mních hrabe zvislo alebo vodorovne (záleží to od štartovacej jeho pozície). Keď narazí na prekážku tak si náhodne vyberie smer, ktorým bude pokračovať (vľavo alebo vpravo). Moje chromozómy nemajú fixné smery kam sa vyberajú po narazení na prekážku. Mních skončí s hrabaním vtedy, keď prejde všetky gény alebo sa zasekol počas hrabania (už sa nemá kam otočiť).
   Fitness vypočítam ako množstvo pohrabaných políčok + počet skál. Počet skál rátam tiež, aby sa mi ľahšie pracovalo s fitness hodnotou. V tomto prípade bude max fitness pre danú záhradu jej obsah vlastný obsah. Keď mám fitness už pridelenú každému jedincovi tak ich následne zoradím podľa nej, aby som mohol jednoducho manipulovať s celou generáciou.

3. **Elitárstvo**
   Na začiatku programu si definujeme aké percento jedincov sa prekopíruje do novej generácie. Sú to tie najlepšie jedince (v tomto prípade jedince s najväčšou fitness).

Počet týchto jedincov nesmie byť dostatočne veľké, aby sme neostali v lokálnom maxime. Elitárstvo zahŕňa 10 percent najlepších jedincov (testovanie mi to ukázalo ako najlepšiu možnosť).

4. **Nová krv**

   Určité percento jedincov, ktoré sa inicializujú v novej generácii (novo vygenerované jedince). Počet týchto jedincov nesmie byť veľa, pretože by sme v podstate len generovali nových jedincov a neuplatňovala by sa podstata generického algoritmu. My chceme aby došlo k evolúcii.

5. **Ruleta a Turnaj**

   Tieto 2 metódy sú zodpovedné za výber rodičov na kríženie. Každá metóda mi vráti 1 rodiča. Dokopy ich mám teda 2.

   Vďaka týmto metódam priradím jedincom s väčšou fitness, väčšiu pravdepodobnosť ich výberu. Avšak niekedy sa môže stať, že nevyberiem tých najlepších. To mi nevadí, pretože aj tie horšie jedince môžu niekedy v sebe skrývať dobré gény.

   Výber jedinca v týchto metódach si predstavme ako tombolu. Každý jedinec dostane určtité množstvo tombolových lístkov. Ťahá sa len raz. Vyhráva ten, koho lístok sa vytiahol.

   Ruletu si predstavme tak, že každý jedinec dostane toľko lístkov do tomboly akú veľkú fitness má. Jedinec A s fitness 70 dostane 70 lístkov, jedinec B s fitness 100 dostane 100 lístkov. Je viac pravdepodobné, že jedinec so 100 lístkami bude ťahaný viackrát. Niekedy sa môže stať, že niektoré jedince budú mať extrémne vysokú fitness a my budeme vyberať iba ich. Potrebujeme to trocha vyvážiť.

   Turnaj si predstavme tak, že každý jedinec dostane toľko lístkov do tomboly ako vysoko v podarí sa nachádza, kde jedince sú zoradené podľa ich fitness. Jedinec A má fitness 70 je na prvom mieste a dostane 1 lístok a jedinec B, ktorý má fitness 100 je na druhom mieste dostane 2 lístky. Takýmto spôsobom dochádza k vyváženiu výberu jedincov za rodičov.

6. **Kríženie**

   V krížení nový jedinec dostane prvú polovicu génov od jedného rodiča druhú od druhého. Tu si dávam pozor, aby sa mi gény neopakovali. Kríženie aplikujem s pravdepodobnosťou 80 percent (testovanie mi to ukázalo ako najlepšiu možnosť). Pravdepodobnosť kríženia nemôže byť moc nízka, pretože to by bolo také elitárstvo, ale nie so silnými jedincami.

```
Jedinec A       8 5 4 7 3 1 6 9 2
Jedinec B       7 2 3 4 5 6 1 2 9
--------------------------------
Nový            8 5 4 7 3 6 1 2 9
```

7. **Mutácia**
   Pri mutácii sa vyberú náhodné gény a vymenia si svoje pozície. V mojom programe môže mutovať 1 chromón až trikrát. Čiže trikrát sa vymenia niektoré gény. Mám to avšak nastavenú náhodnú premennú, ktorá vyberie kolkokrát sa bude mutovať 1,2 alebo 3. Mutáciu aplikujem s pravdepodobnosťou 15 percent (testovanie mi to ukázalo ako najlepšiu možnosť). Mutácia nemôže byť vysoká, inak by dochádzalo k náhodnému generovaniu jedincov.

```
Jedinec          7 2 3 4 5 6 1 2 9
---------------------------------
Nový             6 5 4 7 3 7 1 2 9
```

8. **Nová populácia**
   Keď vykonám všetky hore spomínané kroky na všetkých jedincoch. Tak sa tieto jedince prekopírujú do zoznamu, kde si udržiavam celú populáciu a proces opakujem znovu. Končím vtedy, keď niektorý jedinec dostahne cieľovú fitness. Niekedy sa môže stať, že programu bude dlho trvať, aby sme došli k výsledku a preto som si nastavil, že po 1000 generácii sa evolúcia ukončí a vypíše záhradu pohrabanú najlepším jedincom + graf vývoju fitness hodnoty.

Reprezentácia údajov

Program mám rozdelený do 3 tried. Garden,Chromosome,Evolution.

class Garden:
- height
- width
- garden (dvojrozmené pole, kde zakresľujem objekty a pohrabané políčka)
- number_of_rocks

```
0 0 R 0 0 r
1 1 1 1 1 1
0 0 0 R R 0
0 0 o 0 0 y
```
garden

Pohyb mnícha je označený číslami > 0. Kameň je označený ako R. žltý list ako y, oranžový list ako o, červený list ako r.

class Chromosome:
- garden (objekt triedy Garden)

- fitness
- chromosome (pole, dĺžky polovici obvodu záhrady, kde na každom indexe je číslo vstupu)

```
10 8 5 9 7 3 6 1 4 2
```
chromosome

Čísla reprezentujú v akom poradí vstupuje mních do záhrady. Do záhrady sa vstupuje iba z dvoch strán (vyznačené na obrázku vyššie). Jednotlivé čísla nazývame gény.

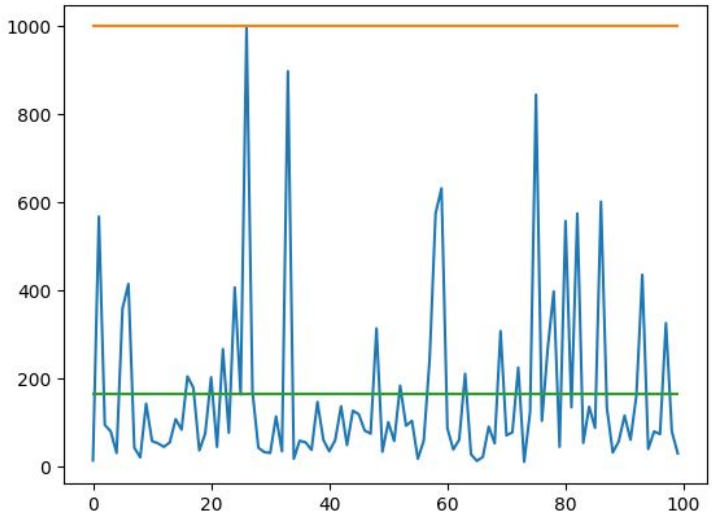
class Evolution:

- number_of_spieces	(počet jedincov v generácii)
- population	(zoznam inštancii chromozómov)
- elite	(koľko percent jedincov sa prekopíruje do ďalšej generácie)
- new_blood	(koľko percent jedincov sa vytvorí v novej generácii)
- cross_percentage	(percentuálna šanca na kríženie)
- mutation_percentage	(percentuálna šanca na mutáciu)

```
[<__main__.Chromosome object at 0x7f24dd3ea050> ...]
```
population

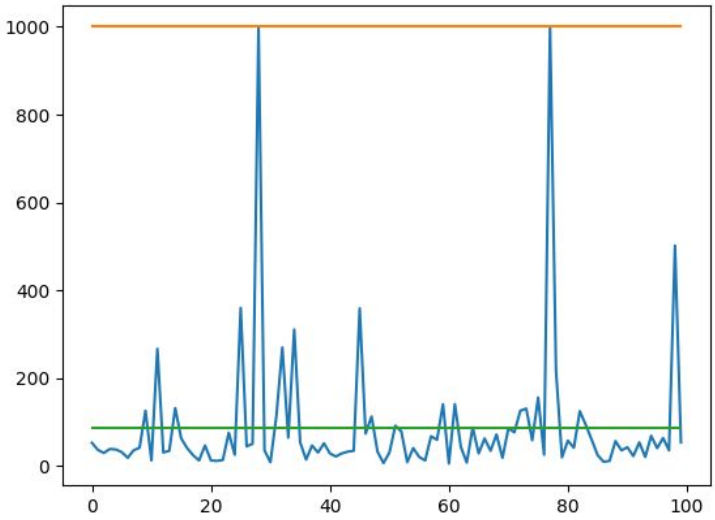
Testovanie

Testovanie som riešil tak, že som si nastavil vstupné hodnoty (počet jedincov, elitárstvo, nová krv, kríženie, mutácia) a následne som to vykonal 100 krát. Potom si vždy ukladal koľko generácii bolo potrebných na vyriešenie úlohy (pri generácii 1000 sám ukončujem)pri daných vstupoch. Následne po zbehnutí testovacej vzorky si mi vykreslí graf kde priemerný počet generácii na vyriešenie úlohy, dané generácie a maximálna hodnota generácie 1000.



Evolution(garden,20,0.05,0.10,0.70,0.10)

Všimol som si, že najlepšie výsledky dostanem, keď zvýšim elitárstvo, kríženie a mutáciu (oproti obrázku hore). Pretože chceme aby sa krížilo viac, tiež potrebujeme trocha viac elitárskych jedincov a nemôžeme to všetko nechať len na krížení tak zdvihneme ak mutáciu.



Evolution(garden,20,0.10,0.10,0.80,0.15)

Ako môžeme vidieť priemerný počet generácii sa znížil + znížil sa aj počet “tŕňov”.

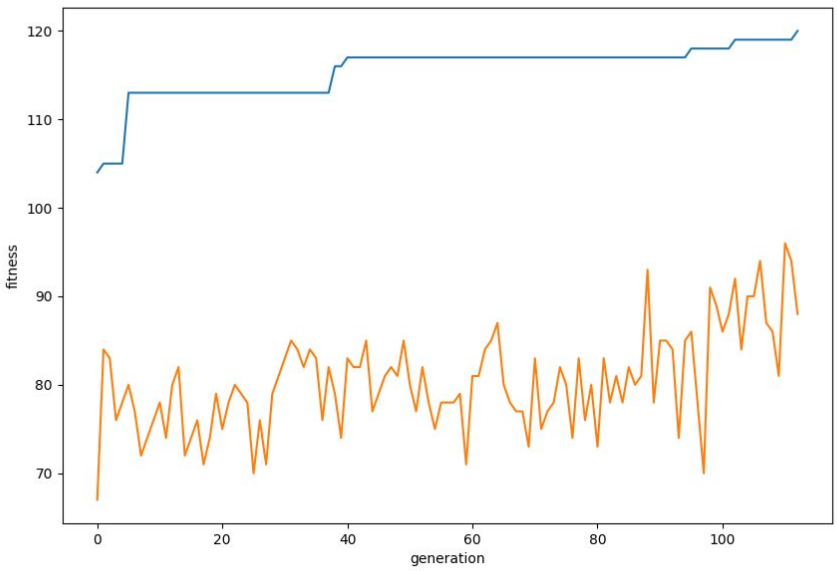
Vzorovné vstupy a výstupy

Vstup

```
10 12
1 5
2 1
3 4
4 2
6 8
6 9
```

Výstup

```
14 18 14  3  4  4  4  3 12 12  2  2
14 18 14  3  4  R  4  3 12 12  2  2
14  R 14  3  4  4  4  3 12 12  2  2
14 14 14  3  R  4  4  3 12 12  2  2
14 14  R  3  4  4  4  3 12 12  2  2
14 14 14  3  4  4  4  3 12 12  2  2
14 14 14  3  3  3  3  3  R  R  2  2
 1  1  1  1  1  1  1  1  1  1  1  1
 6  6  6  6  6  6  6  6  6  6  6  6
 5  5  5  5  5  5  5  5  5  5  5  5
```



Modrá - jedinec s najväčšou fitness
Oranžová - priemerná fitness generácie

Zhodnotenie riešenia a dosiahnutých výsledkov

Program bol navrhnutý za účelom prehľadnosti. S výsledkami algoritmu som spokojný, pretože dokáže pomerne rýchlo nájsť výsledok. Rozmýšľal som aj nad evolučným algoritmom evolučného algoritmu. V podstate by samo prišlo na to ktoré hodnoty pre vstupy sú tie najlepšie.

Používateľská príručka

Program nemá grafické rozhranie, všetky operácie sú vykonávané v termináli.

Program spustíme príkazom **python3 main.py**

Zadáme rozmery záhrady **height width**

Následne zadávame súradnice kameňov **rock_y rock_x**

Nie je potrebné zadať dopredu množstvo kameňov.
Tieto premenné môžeme zadať aj z textového súboru *input.txt* s rovnakým syntaxom.

height width
rock_y rock_x
rock_y rock_x
rock_y rock_x

Potom príkaz v termináli bude vyzerať takto **python3 main.py < input.txt**

Ak chceme výstup zapisovať do súboru output.txt **> output.txt**

Na záver prikladám pomocný súbor *input.txt*, kde je nastavenie záhrady ako na stránke predmetu

Bonus

Bonus je zameraný na pridanie listov 3 rôznych farieb, kde sa musia zbierať v poradí žltý,oranžový,červený. Ak ostávajú farby listov nižšej triedy v záhrade mních nemôže zobrať listy vyššej triedy.

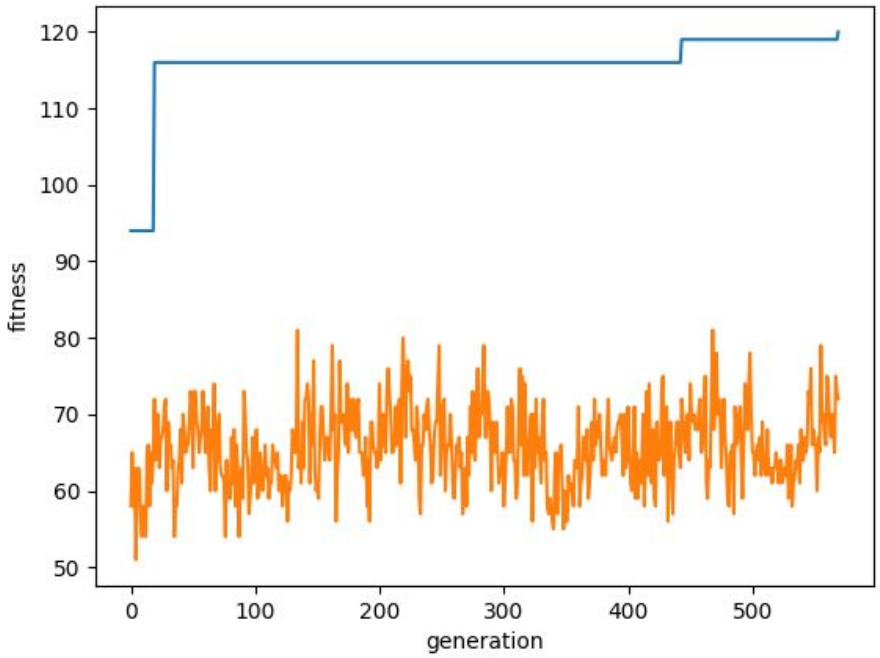
V programe som len trocha upravil funkciu na hrabanie záhrady. Čiže mám 3 počítadla pre dané listy a vždy porovnávam tieto počítadla s listom, na ktorý som natrafil počítalo nižšej triedy signalizuje číslo 0 znamená to, že daný list vyššej triedy môžem pohrabať a naopak.

Vstup

```
10 12
R 0 5
R 0 6
R 9 5
R 9 6
y 4 5
y 4 7
y 5 4
y 5 6
o 2 3
o 2 8
o 3 2
o 3 9
o 5 3
o 6 9
o 7 8
r 9 1
r 0 10
r 1 0
r 1 11
r 6 0
r 7 1
r 8 11
r 9 10
```

Výstup

```
19 17  6  7  3  R  R 22  5 16 20  2
19 17  6  7  3  5  5  5  5 16 16 16
19 17  6  7  3  5  9  9  9  9  9  9
19 17  6  7  3  5  9  9  9  9  9  9
19 17  6  7  3  5  5  5  5  5  5  5
19 17  6  7  3  3  3  3  3  3  3  3
19 17  6  7  8  8  8  8  8  8  8  8
19 17  6  7  8  8  8  8  8  8  8  8
19 17  6  7 12 12 12 12 12 12 12 12
19 17  6  7 12  R  R 18 18 18 18 18
```


120
110
100
90
80
70
60
50
fitness
0
100
200
300
400
500
generation