Fakulta informatiky a informačných technológií

Slovenská technická univerzita

# Zadanie 03
# Evolučný algoritmus dokumentácia

Meno: Samuel Hanák

Cvičenia: pondelok 16:00

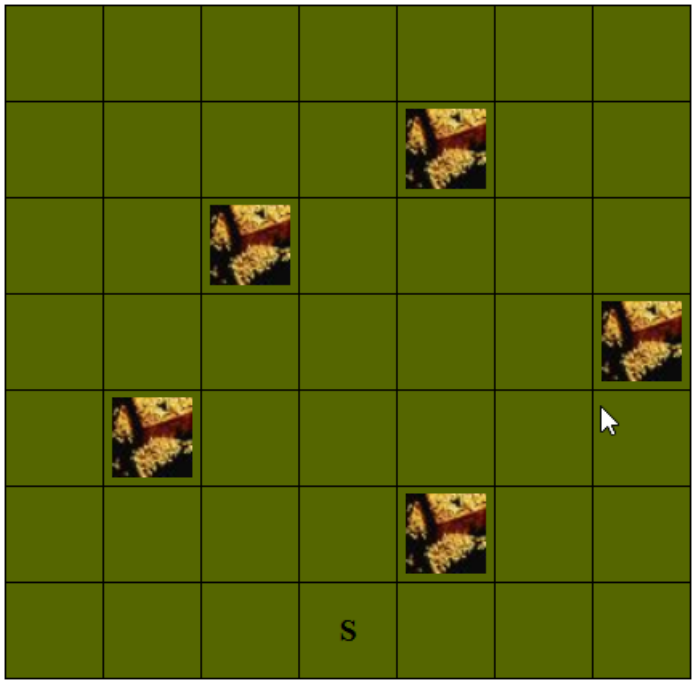
Predmet: Umelá inteligencia

Cvičiaci: Jozef Kováč

Ak. rok: 2018/19

# 1. Zadanie

Majme hľadača pokladov, ktorý sa pohybuje vo svete definovanom dvojrozmernou mriežkou (viď. obrázok) a zbiera poklady, ktoré nájde po ceste. Začína na políčku označenom písmenom S a môže sa pohybovať štyrmi rôznymi smermi: hore **H**, dole **D**, doprava **P** a doľava **L**. K dispozícii má konečný počet krokov. Jeho úlohou je nazbierať čo najviac pokladov. Za nájdenie pokladu sa považuje len pozícia, pri ktorej je hľadač aj poklad na tom istom políčku. Susedné políčka sa neberú do úvahy.




Horeuvedenú úlohu riešte prostredníctvom evolučného programovania nad virtuálnym strojom.

Tento špecifický spôsob evolučného programovania využíva spoločnú pamäť pre údaje a inštrukcie. Pamäť je na začiatku vynulovaná a naplnená od prvej bunky inštrukciami. Za programom alebo od určeného miesta sú uložené inicializačné údaje (ak sú nejaké potrebné). Po inicializácii sa začne vykonávať program od prvej pamäťovej bunky. (Prvou je samozrejme bunka s adresou 000000.) Inštrukcie modifikujú pamäťové bunky, môžu realizovať vetvenie, programové skoky, čítať nejaké údaje zo vstupu a prípadne aj zapisovať na výstup. Program sa končí inštrukciou na zastavenie, po stanovenom počte krokov, pri chybnej inštrukcii, po úplnom alebo nesprávnom výstupe. Kvalita programu sa ohodnotí na základe vyprodukovaného výstupu alebo, keď program nezapisuje na výstup, podľa výsledného stavu určených pamäťových buniek.

# 2. Špecifikácia požiadaviek

Moje riešenie som implementoval pomocou programovacieho jazyku Python 3.6.3 na platforme Windows 10 Pro. Ako vývojové prostredie som použil PyCharm 2017 2.4. Ako prídavné moduly do môjho zadanie som použil json, random a copy.

# 3. Návrh riešenia

## 3.1. Kroky algoritmu

1. Načítaj vstupné hodnoty a vytvor požadovaný počet jedincov, kde naplň prvých 20 buniek náhodnými hodnotami
2. Vlož jedincov do virtuálneho stroja a spracuj výsledky
3. Ak sa našlo riešenie, ukonči program a vypíš cestu
4. V opačnom prípade vyber dvoch jedincov s najlepším fitness a vlož ich do novej generácie (elitarizmus je použitý)
5. Ak je v novej generácií dostatočný počet jedincov prejdi na krok 2.
6. Následne ruletou vyber dvoch jedincov a následne ich so 70% pravdepodobnosťou skríž, inak ich nemodifikuj
7. Následne ich so 7% pravdepodobnosťou mutuj (každého iným typom mutácie) a vlož ich do novej generácie
8. Prejdi na krok 5.

## 3.2. Mutácie

1. Pri prvej mutácií som si vygeneroval dva náhodné čísla buniek a zamenil som ich obsah, pričom som tento princíp zopakoval 30-krát a vrátil zmutovaný gén.
2. Pri druhej som si generoval náhodné čísla buniek a plnil náhodnými hodnotami, kde som tento princíp tak isto zopakoval 30-krát

## 3.3. Kríženie

Najprv som v mojom riešení používal kríženie s jednými pivotom, avšak sa gény v ďalších generáciách príliš podobali a tak som implementoval kríženie s dvoma pivotmi nasledovne:

1. Vygeneruj jedno číslo v rozsahu 0 – 31 a 32 – 63, ktoré budú reprezentovať pivotov
2. Prvé dieťa bude mať gén: mama[0-pivot1]-otec[pivot1-pivot2]-mama[pivot2-64]
3. Druhé dieťa bude mať gén: otec[0-pivot1]-mama[pivot1-pivot2]-otec[pivot2-64]

V testovaní budem používať obidve metódy porovnám výsledky

### 3.4. Selekcia

Ako selekciu som použil ruletu implementovanú podľa nasledujúceho algoritmu:

1. Vypočítaj sumu všetkých fitness hodnôt
2. Vygeneruj náhodné číslo od 0-sumu fitness hodnôt a ulož do S
3. Inicializuj point = 0
4. Zober nespracovaného jedinca a pripočítaj jeho fitness ku point
5. Ak S < point jedinec bol vybratý a algoritmus končí
6. Inak sa vráť na 4.

Algoritmus bude vracať hodnoty s vyššou fitness s väčšou pravdepodobnosťou.

## 4. Testovanie

Každý test bude spúšťať 3-krát aby som dostal, čo najpriemernejšie výsledky. Ako vstup používam rozmiestnenie zo zadania

1. Test case – 5000 generácií, 30 jedincov, kríženie s jedným pivotom

Prvý pokus:

```
Generation 4330
Algorithm found solution:  ['P', 'H', 'L', 'L', 'H', 'L', 'P', 'H', 'P',
```

Druhý pokus:

```
Generation 4468
Algorithm found solution:  ['P', 'H', 'P', 'H', 'P', 'H', 'L', 'L', 'P', 'L',
```

Tretí pokus:

```
Generation 4999
Best fitness this generation:  4
Algorithm haven't found solution, do you want o retry?(yes/no):
```

Nájdené 4 poklady

Zhodnotenie testu: Algoritmus našiel výsledok 2-krát a raz našiel 4 poklady z 5. Môžeme usúdiť, že v priemere by program našiel výsledok do 5000 generácií.

2. Test case – 5000 generácií, 30 jedincov, kríženie s dvoma pivotmi

Prvý pokus

```
Generation 4999
Best fitness this generation:  3
Algorithm haven't found solution, do you want o retry?(yes/no):
```

Nájdené 3 poklady

Druhý pokus

```
Generation 55
Algorithm found solution:  ['H', 'P', 'H', 'P', 'H', 'P', 'H', 'L', 'H', 'P', 'H', 'L', 'L', 'D', 'L', 'D', 'L', 'D', 'L', 'D']
```

Tretí pokus

```
Generation 4999
Best fitness this generation:  4.97
Algorithm haven't found solution, do you want o retry?(yes/no):
```

Nájdené 4 poklady

Zhodnotenie scenáru: Algoritmus podal veľmi odlišné výsledky, kde v jednom prípade našiel iba 3, v druhom skončil s výsledok už v 55 generácií a poslednom našiel 4 poklady. Zhodnocujem, že kríženie s jedným pivotom podáva konzistentnejšie výsledky

3. Test case – 10000 generácií, 50 jedincov, kríženie s jedným pivotom

Prvý pokus

```
Generation 1654
Algorithm found solution:  ['L', 'P', 'L', 'H', 'L', 'P', 'P', 'L', 'H', 'L',
```

Druhý pokus

```
Best fitness this generation:  4.982
Algorithm haven't found solution, do you want o retry?(yes/no):
```

Tretí pokus

```
Generation 125
Algorithm found solution:  ['L', 'H', 'P', 'P', 'L', 'H', 'L', 'L', 'H', 'P', 'P', 'P',
```

Štvrtý pokus

```
Generation 649
Algorithm found solution:  ['P', 'H', 'L', 'L', 'L', 'H', 'P', 'P', 'H', 'P', 'L', 'L',
```

Piaty pokus

```
Generation 163
Algorithm found solution:  ['H', 'P', 'H', 'P', 'H', 'P', 'H', 'L', 'L', 'H', 'L', 'D', 'L', 'D', 'L', 'D']
```

Zhodnotenie: Algoritmus našiel v 4 prípadoch výsledok veľmi rýchlo. V ojedinelom prípade algoritmus výsledok nenašiel, za čo môže asi nevhodne vygenerovaná postupnosť. V tomto prípade by sa po 3000 generáciách postupnosť pregenerovať a algoritmus, by bol konzistentný.

Spravil som test 5- krát, pretože som si nebol istý výsledkami (4:1 je uver hodnejšie ako 2:1)

4. Test case – 5000 generácií, 50 jedincov, kríženie s dvoma pivotmi

Prvý pokus

```
Generation 616
Algorithm found solution:  ['H', 'P', 'H', 'P', 'H', 'P', 'H', 'L', 'H', 'P', 'L', 'L', 'L', 'L', 'D', 'D', 'L', 'P', 'D', 'L']
```

Druhý pokus

```
Best fitness this generation:  4.986
Algorithm haven't found solution, do you want o retry?(yes/no):
```

Tretí pokus

```
Generation 617
Algorithm found solution:  ['L', 'H', 'L', 'H', 'L', 'P', 'H', 'L', 'P', 'H',
```

Štvrtý pokus

```
Generation 928
Algorithm found solution:  ['H', 'P', 'H', 'P', 'H', 'P', 'H', 'L', 'H', 'P', 'H',
```

Piaty pokus

```
Generation 1375
Algorithm found solution:  ['P', 'P', 'H', 'P', 'L', 'L', 'P', 'H', 'P', 'L', 'L', 'P',
```

Zhodnotenie: Algoritmus vrátil výsledky podobne ako pri krížení s jedným pivotom

5. Test case – 2000 generácií, 50 jedincov, kríženie s dvoma pivotmi, pregenerovanie postupnosti pri neúspechu po 2000 generáciách

Prvý pokus

```
Generation 546
Algorithm found solution:  ['H', 'P', 'L', 'L',
```

Nájdené po druhom generovaní

Druhý pokus

```
Generation 712
Algorithm found solution:  ['P', 'H', 'P', 'H', 'P', 'H', 'L', 'H', 'L', 'L', 'P',
```

Nájdené po štvrtom generovaní

Tretí pokus

```
Generation 394
Algorithm found solution:  ['L', 'H', 'H', 'L', 'D', 'D', 'H', 'P', 'P',
```

Nájdené po treťom generovaní

Zhodnotenie: Algoritmu trvalo cca 3-4 minúty do nájdenia riešenia

6. Test case – 2000 generácií, 50 jedincov, kríženie s jedným pivotom, pregenerovanie postupnosti pri neúspechu po 2000 generáciách

Prvý pokus

```
Generation 39
Algorithm found solution:  ['L', 'H', 'L', 'P', 'L', 'P', 'P', 'P', 'H', 'L',
```

Nájdené po 6. generovaní

Druhý pokus

```
Generation 1769
Algorithm found solution:  ['H', 'P', 'H', 'P', 'H', 'P', 'H', 'L', 'H',
```

Nájdené po 2. generovaní

Tretí pokus

```
Generation 738
Algorithm found solution:  ['H', 'L', 'L', 'H', 'P', 'L', 'P', 'L', 'H',
```

Nájdené po prvom generovaní

Zhodnotenie: algoritmu trvalo cca 2 minúty v priemere na nájdenie výsledku

# 5. Zhodnotenie

Algoritmus som testoval v 6-tich scenároch, pričom som zhodnotil, že najlepším riešením je použitie kríženia s jedným pivotom, 50 jedincov a pregenerovanie génov po 2000 generáciách. V riešení mi veľmi zlepšil výsledky elitarizmus, kde som si uchovával najlepších dvoch jedincov.

Myslím, že moje riešenie je použiteľné a dosahuje dobré časové hodnoty.