Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Ilkovičova 2, 842 16 Bratislava 4

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Evolučné programovanie

Adam Šípka

Študijný program: Informatika

Ročník: druhý

Krúžok: Po 14:00

Predmet: Umelá inteligencia

Cvičiaci: doc. Ing. Peter Lacko, PhD.

Ak. rok: 2019/2020

## 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.

## Opis algoritmu

### 1. Inicializácia mapy a prvej generácie

Po spustení programu sa z priloženého súboru *map\_init.ini* načítajú údaje o mape, ako sú jej rozmery, počiatočná pozícia z ktorej začína hľadanie, počet pokladov a ich súradnice. Keď budú dané údaje získané, tak sa môžeme presunúť na ďalšiu časť programu, čo je vytvorenie počiatočnej generácie.

Generácia je reprezentovaná cez *nested dictionary*, kde sú ako kľúče požité hodnota fitness, postupnosť krokov, čo je vlastne sekvencia pamäťových buniek, ktorá sa bude vykonávať, cesta akou sa pohyboval a nájdené poklady. Pri tvorení prvej generácie sú všetky počiatočné hodnoty nulové, až na sekvenciu pamäťových buniek, kde je prvá polovica naplnená náhodnými údajmi, čo sú čísla od 0 do 255, keďže pamäťová bunka je 8-bitová. Počet jedincov v  generácii je 100, čo je ideálne množstvo, vďaka ktorému dokážeme spoľahlivo a zároveň rýchlo nájsť riešenie.

### 2. Vykonanie programu, nájdené poklady a fitness

Keď je prvá generácia vytvorená, tak môžeme začať s vykonaním programu. Prebieha v cykle, v ktorom najprv virtuálny stroj vykoná program, následne zistíme ako sa jedinec na mape pohyboval, či z nej vystúpil, alebo našiel nejaké poklady a pod. Nakoniec vyrátame fitness aktuálneho jedinca a presunieme sa na ďalšieho. Teraz to rozoberiem podrobnejšie.

Virtuálny stroj je reprezentovaný funkciou *execute()*, do ktorej ako parameter vkladáme aktuálneho jedinca. V nej, vo *while* cykle, ktorý je prerušený ak sa dostaneme na poslednú pamäťovú bunku, alebo ak počet vykonaných operácií presiahne 500, prebieha vykonávanie programu. Ako prvé si vytvoríme kópiu jedinca, keďže samotná sekvencia pamäťových buniek musí kvôli kríženiu ostať v pôvodnom stave. Z prvej bunky zoberieme prvé dva bity (operátor) a zvyšných 6 (adresa, na ktorej bude operácia vykonaná). Použil som len štyri inštrukcie, ktoré boli uvedené v zadaní (inkrementácia, dekrementácia, skok a výpis). Pri výpise je do zoznamu, v ktorom je uložená pôvodná cesta, akou sa jedinec na mape hýbe, vložený znak, ktorý označuje jeho smer. Smer pohybu som získaval z posledných dvoch bitov, pričom hodnoty boli reprezentované nasledovne: 11 vpravo **P**, 10 vľavo **L**, 01 dole **D** a 00 hore **H**. Program sa teda vykonával až dovtedy, pokým nebol počet vykonaných operácii 500, alebo nedosiahol poslednú pamäťovú bunku.

Po funkcii *execute()* je spustená funkcia *found\_treasures()*, kde zo získanej postupnosti pohybov získame finálnu cestu, ktorou sa jedinec hýbal v hraniciach mapy, keďže pôvodná môže vystupovať mimo ňu. Začíname od počiatočných súradníc a podľa smeru, v akom sa hýbeme, ich meníme. Po každom posune skontrolujeme, či je nová súradnica stále na mape, ak náhodou nie je, tak funkciu prerušíme. Po kontrole pridáme smer pohybu do finálnej cesty a ideme skontrolovať, či sa na danej súradnici nenachádza jeden z pokladov. Ak sa nachádza, a ak to je poklad na ktorého súradniciach sme ešte neboli, tak ho pridáme do zoznamu nájdených pokladov. Postupne takýmto spôsobom prejdeme celú pôvodnú cestu, pokiaľ nevystupuje z mapy, alebo nenašla všetky poklady.

Ak máme finálnu cestu hotovú a získané informácie o nájdených pokladoch, tak sa presúvame k zisteniu hodnoty fitness. Na to je určená funkcia *calculate\_fitness()*. Existuje viacero spôsobov jeho rátania a podľa predtým zistených informácii vyberieme ten správny. Môže nastať prípad, že finálna cesta má dĺžku 0, čiže neexistuje. V tomto prípade je aj fitness daného jedinca nastavený na nulu. Ak cesta existuje, no poklady neboli nájdené, tak fitness hodnota nemôže presiahnuť 1. Vtedy ju vyrátame tak, že vydelíme dĺžku cesty veľkosťou mapy (stĺpce x riadky) a ak je výsledná hodnota väčšia, alebo rovná 1, tak ju delíme desiatimi, pokým nebude výsledok prijateľný. Posledný spôsob zisťovania fitness je ten najdôležitejší, keďže v predchádzajúcich dvoch som sa venoval prípadom, v ktorých neboli nájdené žiadne poklady. Teraz je hlavnou zložkou rátania fitness-u počet nájdených pokladov a je zariadené, aby jedinci s väčším počtom nájdených pokladov mali vždy väčšiu hodnotu, ako jedinci s menším počtom. Ale hodnota fitness závisí aj od dĺžky cesty, lepšia je krajšia. Ak je cesta kratšia ako veľkosť mapy, tak vzorec na fitness je nasledovný:

*fitness = počet nájdených pokladov + (1 – (dĺžka cesty / veľkosť mapy))*

Ak je cesta dlhšia, ako počet všetkých políčok, tak postupujeme podobne ako v predchádzajúcom prípade, delíme desiatimi dovtedy, pokým hodnota nie je menšia ako jedna. Ale aj vtedy môže dôjsť k prípadu, v ktorom by bola uprednostnená dlhšia cesta pred kratšou, takže výslednú hodnotu vydelíme desiatimi ešte jeden krát. Potom k nej ešte prirátame počet nájdených pokladov a fitness máme vyrátaný.

### 3. Tvorba novej generácie

Po vykonaní predchádzajúceho kroku pre všetkých jedincov v danej generácií je potrebné vytvoriť novú, ale ešte predtým sú jedinci zo starej generácie zoradený podľa ich fitness hodnôt. Na to slúži funkcia *sort\_individuals()*, ktorá nám vráti zoznam obsahujúci indexy jedincov zoradených od najvyššej fitness hodnoty po najnižšiu. Potom ešte funkcia *generation\_summary()*, vypíše prehľad o vykonanej generácii, kde sú vypísané fitness, počet nájdených pokladov a dĺžka cesty najlepšieho jedinca, ako aj celkový priemer týchto hodnôt.

Následne sa môžeme presunúť na tvorbu novej generácie. V programe som zaviedol elitárstvo, čiže určitá časť jedincov z predchádzajúcej generácie je automaticky a bez úprav presunutá do novej. Skúšal som rôzne hodnoty a vďaka testovaniu som prišiel na to, že počet najlepších jedincov, ktorí majú byť súčasťou novej generácie, je okolo 2 až 5%, (v programe mám použité 2) lebo pri vyšších hodnotách pomaly začínajú prevažovať nedostatky elitárstva.

Ďalšia časť jedincov je vytvorená náhodne, tak ako pri prvej generácii, keďže je dobré mať istý podiel nových jedincov, vďaka ktorým nebude vývoj určený len pôvodnou generáciou, ale budú sa v ňom vyskytovať aj nové, náhodné prvky, ktoré môžu poskytnúť väčšiu variáciu výsledkov. Percento nových jedincov by tiež nemalo byť príliš vysoké, okolo 15% je vhodná hodnota, keďže čím vyšší je počet nových náhodných prvkov, tak aj celkový vývoj je čoraz viac náhodný.

Zvyšných jedincov získame pomocou kríženia dvoch jedincov z predchádzajúcej generácie. Na to nám poslúži predtým získaný zoznam indexov zoradených podľa fitness hodnoty. Postupne vyberáme dvoch najlepších a vo funkcii *crossover()* ich skrížime a dostaneme nového jedinca. Kríženie prebieha tak, že postupne striedame vyberanie hodnôt z rodičov, z jedného vyberáme len párne prvky a z druhého tie nepárne. Pri týchto jedincoch je aj šanca mutácie, aby mohlo dôjsť k obmene niektorých jedincov. Nie je dobré mať príliš vysokú šancu na priebeh mutácie, nová generácia by mohla mať príliš veľa náhodných jedincov, ale záleží aj na type mutácie. V programe mám použité dva typy, pri jednom mutuje len jedna pamäťová bunka, pri druhom náhodný počet pamäťových buniek.

Na to, aby bola použitá mutácia, musia byť splnené nasledujúce podmienky. Najprv vyberieme náhodné číslo medzi 0 a 1. Potom porovnávame SINGLE\_CELL\_MUTATION a náhodné číslo, ak je SINGLE\_CELL\_MUTATION väčšie, tak prebehne mutácia minimálne jednej pamäťovej bunky. Náhodné číslo ešte porovnáme s MULTIPLE\_CELL\_MUTATION, ktoré má nižšiu hodnotu, ako jeho predchádzajúci variant. Ak bude aj to väčšie, zvolí sa náhodné číslo z intervalu 0 až 64 (kvôli počtu pamäťových buniek), ktoré symbolizuje koľko krát sa funkcia *mutation()* vykoná. V nej zvolíme náhodnú pamäťovú bunku a jej hodnotu nahradíme novou, náhradnou hodnotou. Ak bude ale číslo menšie, tak sa hneď presunieme na funkciu *mutation()*, ktorá bude vykonaná len raz.

## Testovanie a porovnanie výsledkov

V programe som pri testovaní skúšal meniť stavy niektorých parametrov, ako je napríklad percento najlepších jedincov, ktorí sa automaticky presunú do nasledujúcej generácie, alebo rôzne typy mutácii a šance na ich výskyt, a priraďoval som im rôzne hodnoty. V každej generácii som pozoroval priemernú hodnotu fitness a zaznamenával som si ju do zoznamu, ktorý bol po skončení programu načítaný do .cvs súboru. Ten som následne exportoval do excelu a vytvoril som z neho doleuvedené grafy. Pre každý prípad sú uvedené dva grafy, jeden obsahuje výsledky viacerých behov programu, keďže graf len jedného behu nie je objektívny a v iných behoch by mohli byť nadobudnuté priveľmi rozdielne hodnoty.  Ďalší graf, kvôli zvýšeniu celkovej prehľadnosti, obsahuje priemer predchádzajúcich hodnôt. Každá generácia sa skladala zo 100 jedincov a  počet vykonaných generácií bol nastavený na 1500.

Do grafov som zaznamenával len priemerný fitness a nie fitness najlepšieho dosiahnutého výsledku, keďže vo väčšine prípadov boli nájdené všetky poklady a hodnoty v takýchto grafoch by neboli príliš zaujímavé na analyzovanie, lebo sa menili len pár krát za beh. Hodnoty, ktoré som menil, aby som mohol pozorovať zmenu výsledkov, boli percento elitárstva, skúsil som 2% a 10%. Taktiež som menil šancu na výskyt mutácie. Pre typ mutácie, pri ktorom mutuje väčší počet pamäťových buniek, som nastavoval hodnotu na 30%, alebo 10% a v jednom prípade boli použité oba typy, šanca mutácie jednej pamäťovej bunky bola 30%, šanca na väčší počet buniek bola 10%. Podrobnejší opis všetkých jednotlivých scenárov a zhodnotenie nameraných výsledkov je uvedené nižšie, pod príslušnými grafmi.

V prvom prípade boli pri tvorbe nasledujúcej generácie použité 2% najlepších jedincov z minulej, počet nových bol 15% a bola použitá mutácia viacerých pamäťových buniek s 10% šancou výskytu. Priemerná hodnota fitness sa pohybovala v rozmedzí od 0,6 až do 1,6 (horný graf), no pre každý jednotlivý beh, nebol rozdiel medzi maximom a minimom až taký veľký, jeho hodnota bola približne 0,5. Počas prvých 200 generácii priemerný fitness prudko stúpal, až dosiahol určitú hranicu a následne sa pohyboval medzi 0,9 a 1,2, čo značí, že v priemere takmer každý jedinec našiel aspoň jeden poklad (dolný graf).

V druhom prípade bola oproti prvému zmenená len hodnota elitárstva, nastavil som ju na 10%. Ako je na oboch grafoch vidieť, tak počas prvých 200 generácií fitness rapídne stúpal a dosiahol hodnoty od 3 až po 3,7 a následne naberal na hodnote ešte viac a v priemere sa udržoval tesne pod hranicou štvorky. Vysoké percento elitárstva niekoľkonásobne navýšilo priemerný fitness, ale nájdenie všetkých pokladov trvalo dlhšie ako v predchádzajúcom scenári, keďže väčšina novovytvorených jedincov bola zameraná len na nájdenie tých istých, už predtým získaných pokladov a hľadať nové sa príliš nezberala.

Keď v programe bola využitá mutácia náhodného počtu pamäťových buniek a šanca na jej výskyt bola nastavená na 30%, tak môžeme vidieť, že fitness sa pri všetkých behoch pohyboval medzi 0,4 a 1, čo sú jeho minimálne a maximálne hodnoty, no v celkovom priemere bol v rozmedzí od 0,6 po 0,8. V hornom grafe je vidieť, že vysoké percento výskytu mutácie nám zvyšuje náhodnosť dosahovaných výsledkov. Vidíme, že fitness stúpal len na začiatku behu programu, prešlo pár prvých generácii a jeho hodnota bola odvtedy viac menej konštantná, nie je vidieť jej celkový nárast, alebo pokles.

V poslednom prípade boli použité oba spôsoby mutácie, šanca na výskyt mutácie jednej pamäťovej bunky bola nastavená na 30%, šanca na mutáciu viacerých bola na 10%. V grafe môžeme vidieť, že menej úspešné behy boli v rozmedzí približne od 0,6 po 1,2, zatiaľ čo tie lepšie boli medzi 1,2 až 1,7. 30% šanca na mutáciu jednej bunky nemá negatívne dôsledky, aj keď jej hodnota je celkom vysoká. Keďže mutuje len jedna bunka, celkový výsledok programu sa nebude príliš odlišovať od pôvodného a vďaka nemu sú výsledky o trochu lepšie ako v prvom scenári, v ktorom je len jeden spôsob mutácie.

## Zhrnutie a zhodnotenie riešenia

V programe mám pri tvorbe nových generácií zavedené elitárstvo, vďaka ktorému sa isté percento najlepších jedincov automaticky dostane do nasledujúcej generácie, takže o nich neprídeme. Aby nedošlo k stagnácií vývoja, tak je istá časť novej generácie vytvorená nanovo a náhodne. Zvyšné jedince sa medzi sebou krížia, postupne sa striedajú pamäťové bunky z oboch rodičov a vytvorí sa nový jedinec. Okrem toho mám zavedené dva spôsoby mutácie, jeden mení náhodný počet pamäťových buniek jedinca a ďalší len jednu. Druhý spôsob má samozrejme väčšiu šancu na výskyt, keďže nemá na daného jedinca až tak veľký dopad.

Jeden beh programu, ak bol počet generácií 1500 a počet jedincov v nej 100, trval 1 minútu a 25 sekúnd. Riešenie bolo väčšinou nájdené medzi 500. a 1000. generáciou, ak boli použité parametre z posledného scenára. Na väčšiu efektivitu je možné správne meniť počet jedincov, taktiež prenastaviť niektoré parametre, napríklad znížiť, alebo zvýšiť elitárstvo, podiel nových jedincov, šancu na výskyt mutácií a ak je všetko správne, tak aj mierne zredukovať počet generácií. Avšak správne určiť tieto hodnoty a nájsť ich optimálny stav je zložité. Ale ak by sa to podarilo, riešenie by mohlo byť nájdené aj pri menšom počte generácií s nižším počtom jedincov.