Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 812 19 Bratislava

Dokumentácia k zadaniu z UI:

Zenová záhrada

Autor: **Peter Markuš**
Študijný program: INFO-4
Akademický rok: 2016/2017

Obsah

1 Zadanie úlohy

Zenová záhradka je plocha vysypaná hrubším pieskom (drobnými kamienkami). Obsahuje však aj nepohyblivé väčšie objekty, ako napríklad kamene, sochy, konštrukcie, samorasty. Mních má upraviť piesok v záhradke pomocou hrablí tak, že vzniknú pásy ako na nasledujúcom obrázk.

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

Pásy môžu ísť len vodorovne alebo zvislo, nikdy nie šikmo. Začína vždy na okraji záhradky a ťahá rovný pás až po druhý okraj alebo po prekážku. Na okraji - mimo záhradky môže chodiť ako chce. Ak však príde k prekážke - kameňu alebo už pohrabanému piesku - musí sa otočiť, ak má kam. Ak má voľné smery vľavo aj vpravo, je jeho vec, kam sa otočí. Ak má voľný len jeden smer, otočí sa tam. Ak sa nemá kam otočiť, je koniec hry. Úspešná hra je taká, v ktorej mních dokáže za daných pravidiel pohrabať celú záhradu, prípade maximálny možný počet políčok. Výstupom je pokrytie danej záhrady prechodmi mnícha. Pokrytie zodpovedajúce presne prvému obrázku (priebežný stav) je napríklad takéto:

Uvedenú úlohu riešte pomocou evolučného algoritmu. Maximálny počet génov nesmie presiahnuť polovicu obvodu záhrady plus počet kameňov, v našom príklade podľa prvého obrázku 12+10+6=28. Fitnes je určená počtom pohrabaných políčok. Výstupom je matica, znázorňujúca cesty mnícha. Je potrebné, aby program zvládal aspoň záhradku podľa prvého obrázku, ale vstupom môže byť v princípe ľubovoľná mapa.

2 Implementácia

2.1 Dátové štruktúry a ich funkcie

Controller – slúži ako spúšťač programu. Pri jeho vytvorení sa zo súboru načíta mapa, slúžiaca ako argument pre následne zavolanú triedu EvoluteSearch, ktorá pre ňu vyhľadá riešenie.

Coordinate – reprezentuje pohrabávané políčka po mape. Obsahuje riadkovú, stĺpcovú a taktiež predchádzajúcu súradnicu. Okrem súradníc si pamätá aj smer, prostredníctvom ktorého postupuje pohrabávanie postupne.

EvoluteSearch – rieši problém Zenovej záhrady prostredníctvom hlavnej metódy **genetic**, ktorá je opísaná v časti opisu algoritmu. Obsahuje atribúty ako inštranciu záhrady, počet génov a populáciu vo forme 2d pola. Okrem týchto atribút obsahuje aj rôzne parametre pre možnosť modifikácie spôsobu vyhľadávania riešenia. Tieto atribúty sú v zdrojovom kóde a v dokumentácii označené kapitálkami.
Funkcie:
generateChromozome() – vracia pole veľkosti počtu génov v ktorom boli vygenerované náhodné čísla od 1 po obvod mapy, pričom majú 50% pravdepodobnosť, že budú záporné. Týmto spôsobom vytvoríme jedincov s náhodne nastavenými génmi.
doTournament(int[]) – vygeneruje 2 alebo 3 (záleží na parametri TOURNAMENT_SELECT) náhodné čísla od 1 po POP_SIZE (100), ktoré slúžia ako indexy pre pole *fitness* (ktorému zodpovedajú aj indexy z pola population) obsahujúci počet pohrabanej plochy. Funkcia prostredníctvom tohto pola vybere najúspešnejší z vybratých jedincov.

Garden – trieda uchovávajúca mapu záhrady vo forme 2d pola. Ako ďalšie atribúty obsahuje aj jej vlastnosti ako počet potrebných pohrabaní, počet stĺpcov, riadkov, kameňov a dĺžku poloobvodu.
Funkcie:
Garden(File) – slúži na parsovanie súboru pre vytvorenie mapy a výpočet jej vlastností
rakeGarden(int[], boolean) – funkcia na pohrabávanie záhrady. Prostredníctvom funkcie **copyMap** sa vytvorí kópia novej mapy, do ktorej budeme zaznačovať aktuálne pohrabávanie.
{ Na základe parametra pola chromozóm sa najprv pre každý gén špecifikuje štartovacia súradnica prostredníctvom funkcie **getDirection**. Predtým, ako začne slučka postupného pohrabávania sa najprv skontroluje, či sa na štartovacej súradníci nenachádza kameň alebo už pohrabaný piesok.
{ V následnej podmienke novej slučky sa nachádza funkcia **inMapBounds,** ktorá zabezpečí trvanie pohrabávania dovtedy, dokým sa pohrabávač nebude nachádzať na konci záhrady.
Slučka začína ďalšou kontrolou, či sa na aktuálnej súradníci nenachádza kameň označeným číslom -1 alebo už pohrabaný piesok, ktorý je označený poradovým číslom pohrabávania.

- ak áno, zavoláme funkciu **changeDirection,** do ktorej dáme ako parameter aktuálnu súradnicu (narazenú) a vráti ju nám upravenú tak, aby zmenila smer a nebola narazená. Ak sa taký smer nenájde, v ďalšej slučke vymažeme celú cestu tak, že označíme pohrabané políčka ako nepohrabané prostredníctvom predchádzajúcich súradníc.

Nasleduje značenie do záhrady poradovým číslom pohrabávania. Spočítaním aktuálnych súradníc a smerom s ktorým korešpondujú, vytvoríme nové súradnice. **}**
Na konci každého úspešného pohrabávania sa inkrementuje poradové číslo a začína ďalší gén. **}** Po vykonaní všetkých pohrabávaní jedným jedincom sa skontroluje ďalší parameter funkcie *print*, ktorý keď je true, zavolá funkciu **printMap** na výpis riešenia.
Na konci funkcia vráti počet pohrabaného piesku prostredníctvom funkcie **countRakedSand.**

getDirection(int) – na základe parametra, ktorým je číslo génu, sa určí smer a počiatočná súradnica na okraji záhrady, od ktorej sa začne pohrabávať. Nasledujúci príklad vysvetľuje ako toto určovanie funguje prostredníctvom štyroch podmienok.

C = 12	(#columns)	dirC	(direction column)
R = 10	(#rows)	dirR	(direction row)
HP = 22	(half perimeter)		
CH = (44,44)	(gene number)		

KONČIACA SÚRADNICA (druhú súradnicu nieje potreba, lebo sa pohybujeme len jedným smerom)
POČIATOČNÁ SÚRADNICA
SMER

CH(10) <= S(12)

R=0	C=CH-1	dirC=1	dirR=0	>
0	**9**	**1**	**0**	

CH(15) <= PO(22)

R=CH-S-1	C=S-1	dirC=0	dirR=-1	^
2	**11**	**0**	**-1**	

CH(25) <= PO+R (32)

R=CH-HP-1	C=0	dirC=0	dirR=1	v
2	**0**	**0**	**1**	

CH > PO+R(32)

R=R-1	C=CH-HP-R	dirC=-1	dirR=0	<
9	**3**	**-1**	**0**	

changeDirection(Coordinate, int, int[][]) – funkcia zabezpečujúca zmeny smeru v prípadoch kolízie. Ako parameter má súradnicu (narazenú), ktorej je potrebné zmeniť smer, číslo génu, ktorý viedol ku kolízii a 2d pole mapy.
Na začiatok vytvorí novú predchádzajúcu (ešte nenarazenú) súradnicu a zistí, či sa jedná o smer vertikálny alebo horizontálny. Podľa toho sa vytvorí nová susediaca súradnica so zmeneným smerom, ku ktorej je možné pristúpiť (je tam nepohrabaný piesok). Ak sú také súradnice dve, gén rozhodne kadiaľ pôjde na základe jeho záporného alebo kladného čísla. Ak žiadny smer nenájde, vráti súradnicu null.

2.2 Opis algoritmu

Algoritmus začína inicializovaním chromozómov do 2d pola populácie prostredníctvom funkcie **generateChromozome**. Následne začína hlavná slučka programu ktorá prebehne najvyšším možným počtom GENERATIONS (10000).

1. { Pre každého jedinca zavoláme funkciu **rakeGarden**, ktorá bude na základe jeho chromozómu pohrabávať záhradu a vypočíta, koľko plochy sa mu podarilo pokryť do fitness pola. (podrobnejší opis je vo funkcii rakeGarden)
2. Vo fitness poli nájdeme najúspešnejšieho jedinca a zapíšeme jeho hodnotu do *max* a jeho index pozíciu do *iMax* a vypíšeme koľkatá generácia sa spracúva spolu s hodnotami *max* a *mutRateAct* (vysvetlený nižšie).
3. Ak hodnota max sa rovná počtu potrebných pohrabaní v záhrade, našlo sa riešenie a pre populáciu s indexom chromozómu *iMax* sa zavolá **rakeGarden** spolu s parametrom true na výpis mapy s riešením. Vypíše sa taktiež postupnosť všetkých génov chromozómu.
4. Ak sme riešenie ešte nenašli, hodnota *mutRateAct* ak nieje väčšia ako MAX_MUT_RATE (30%), tak sa inkrementuje o 0.01, v opačnom prípade sa nastaví na MIN_MUT_RATE (5%). Tieto hodnoty ovplyvňuje následné vytváranie novej populácie.
5. { zdeklarujeme 2 čísla *individual* ktoré budú reprezentovať najúspešnejší index chromozómu z 2 alebo 3 náhodne vybraných jedincov pomocou funkcie **doTournament.** Tie potom postupne skopírujeme do nového 2d pola *children.*
6. Následne s CROSS_RATE (90%) pravdepodobnosťou sa bude diať kríženie.
   { Pre každý gén z týchto dvoch indexov v poli *children* nastane mutácia s pravdepodobnosťou *mutRateAct* (2-30%)
7. Vygeneruje sa náhodné číslo *mutNum* od 1 po obvod pri ktorom sa ešte s 50% pravdepodobnosťou zmení na záporné číslo. Toto číslo sa bude hľadať vo všetkých génoch chromozómu aktuálneho indexu pola *children*.
8. Ak sa nájde, uložíme si toto číslo génu a vymeníme ho s aktuálne vybratým génom, čo zabezpečí odlišné poradie pohrabávania. Ak sa nenájde, tak aktuálne vybraný gén prepíšeme génom s číslom *mutNum*. } }
9. Po skončení vytvárania potomkov si pre nasledujúcu generáciu zapíšeme najúspešnejšieho jedinca do prvého indexu pola *children*. Následne staré pole population prepíšeme polom *children*. }

3 Zhodnotenie

Implementácia bola zrealizovaná v programovacom jazyku Java. Je potrebné použiť kompilátor JDK/JavaSE s verziou 1.8. Algoritmus hľadania používa 6 rôznych nastaviteľných parametrov. Výhody, nevýhody (nároky na pamäť, časová zložitosť)

TOURNAMENT_SELECTION – zahŕňa 2 rôzne metódy selekcie, jedna vyberá z dvoch rôznych jedincov, druhá z troch. Druhá metóda je vždy výhodnejšia.

GENERATIONS – čím väčší počet generácií, tým väčší máme dostupný čas pre vyhľadanie riešenia.

POP_SIZE – čím je väčšia populácia, tým pomalšie pokročujú generácie. Celková zvýšena časová zložitosť môže nastať v prípade neoptimálneho pomeru medzi počtom mutácii a počtom populácie. Námet na vylepšenie algoritmu spočíva v nájdený optimálnejšieho pomeru akým je aktuálny. Ďalším možným vylepšením je nájsť najvýhodnejšie hranice hodnôt MIN_MUT_RATE a MAX_MUT_RATE.

CROSS_RATE – pre optimálnejšie riešenie by mohlo obsahovať hodnotu 1 pre 100% šancu mutovania.