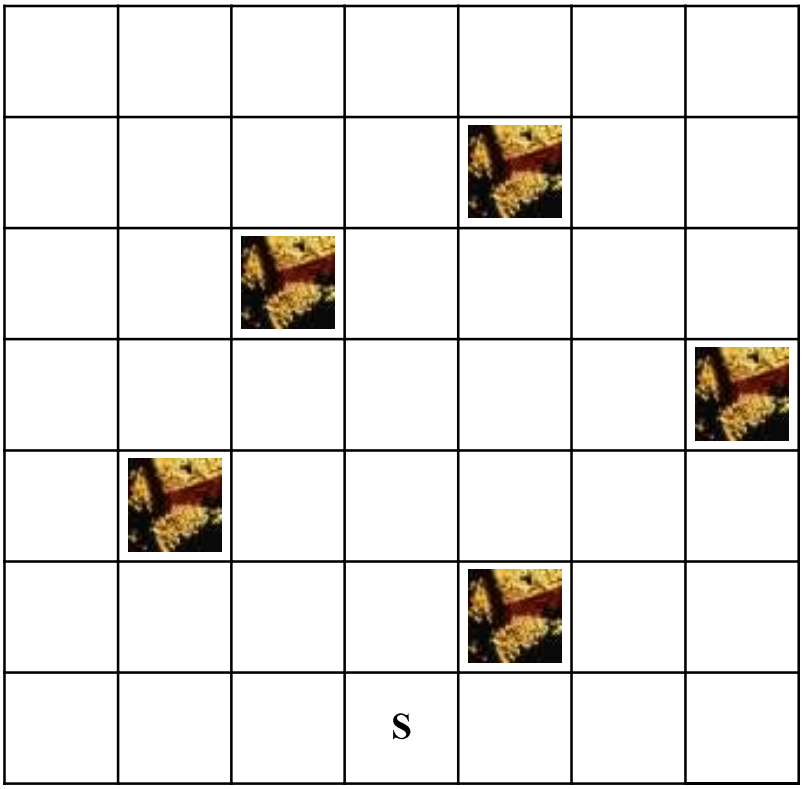
Hľadanie pokladu

Zadanie č. 3b

Úloha

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.



Zadanie

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.

Virtuálny stroj

Náš stroj bude mať **64 pamäťových buniek** o veľkosti **1 byte**.

Bude poznať štyri inštrukcie: inkrementáciu hodnoty pamäťovej bunky, dekrementáciu hodnoty pamäťovej bunky, skok na adresu a výpis (**H**, **D**, **P** alebo **L**) podľa hodnoty pamäťovej bunky. Inštrukcie majú tvar podľa nasledovnej tabuľky:

inštrukcia	tvar
inkrementácia	00XXXXXX
dekrementácia	01XXXXXX
skok	10XXXXXX
výpis	11XXXXXX

Hodnota XXXXXX predstavuje 6-bitovú adresu pamäťovej bunky s ktorou inštrukcia pracuje (adresovať je teda možné každú). Prvé tri inštrukcie by mali byť jasné, pri poslednej je potrebné si dodefinovať, čo sa vypíše pri akej hodnote bunky. Napríklad ak bude obsahovať maximálne dve jednotky, tak to bude **H**, pre tri a štyri to bude **D**, pre päť a šesť to bude **P** a pre sedem a osem jednotiek v pamäťovej bunke to bude **L**. Ako ukážku si uvedieme jednoduchý príklad:

Adresa:	000000	000001	000010	000011	000100	000101	000110	**...**
Hodnota:	00000000	00011111	00010000	01010000	00000101	11000000	10000100	**...**

Výstup tohoto programu bude postupnosť: P H H H D H ... Ďalšie hodnoty budú závisieť od hodnôt nasledujúcich pamäťových buniek. (Dúfam, že je jasné, že uvedenú postupnosť vypisuje inštrukcia v pamäťovej bunke s adresou 5. A program je samomodifikujúci sa, takže vypísané hodnoty nezodpovedajú hodnotám na začiatku, ale počas behu programu.) **Sú možné (a odporúčané) aj lepšie reprezentácie hodnôt pre H D P a L, napríklad podľa posledných dvoch bitov**.

Program sa zastaví, akonáhle bude splnená niektorá z nasledovných podmienok:

1. program našiel všetky poklady
2. postupnosť, generovaná programom, vybočila zo stanovenej mriežky
3. program vykonal 500 krokov (inštrukcií)

Či sa program zastaví, keď príde na poslednú bunku alebo pokračuje znovu od začiatku, si môžete zvoliť sami. (Môžete to nechať aj na voľbu používateľovi.)

Je možné navrhnúť aj komplikovanejší virtuálny stroj s rozšírenými inštrukciami a veľkosťou pamäťovej bunky viac ako osem bitov, ale musí sa dodržať podmienka maximálneho počtu

pamäťových buniek 64 a limit 500 krokov programu. Rozšírenie inštrukcií sa využíva nielen na zadefinovanie nových typov inštrukcií, ale hlavne na vytvorenie inštrukcií s podmieneným vykonávaním.

Zaujímavou (a fungujúcou) obmenou je aj použitie nepriamej adresácie. Vtedy predstavuje hodnota XXXXXX adresu bunky, kde je (v posledných šiestich bitoch) uložená adresa bunky ktorá sa má modifikovať alebo ktorá sa číta. Pri inštrukcii skoku je to adresa bunky kde je uložená adresa skoku. Nepriama adresácia sa dá využiť na oddelenie programu a údajov (napríklad keď z nejakého dôvodu nechceme vytvoriť samomodifikujúce sa programy). To je však pri tomto spôsobe tvorby programov len málokedy výhodné.

Evolučný algoritmus

Hľadanie riešenia prebieha podľa nasledovného postupu:

1. Zo vstupného súboru sa načíta rozmer mriežky, štartovacia pozícia, počet a rozmiestnenie pokladov.
2. Počiatočná populácia jedincov (najmenej 20) sa nainicializuje náhodnými hodnotami v stanovenom rozsahu (napríklad prvých 16 alebo viac buniek každého jedinca).
3. Každému jedincovi sa určí fitness – počet nájdených pokladov do zastavenia jeho programu.
4. Ak je nájdený jedinec, ktorý našiel všetky poklady, riešenie končí s úspechom a vypísaním programu a postupnosti, ktorú vygeneroval. Ak sa vytvoril požadovaný počet populácií, algoritmus vypíše doteraz nájdené najlepšie riešenie a čaká na rozhodnutie používateľa – koniec alebo ďalšie opakovanie.
5. Jednou z metód selekcie (ruleta, turnaj a iné) sa určia rodičia a krížením vytvoria nových potomkov.
6. Nové jedince s istou pravdepodobnosťou mutujú a vstupujú do novej populácie.
7. Keď je nová generácia kompletná, prejde sa na vykonávanie kroku 3.

Hodnota fitness závisí v prvom rade od počtu nájdených pokladov, ale je vhodné ju zjemniť podľa počtu vykonaných krokov – kratšia postupnosť je lepšia.

Tu je ukážka, ako sa mení pravdepodobnosť výberu zvoleného jedinca od počtu jedincov v turnaji. Pri dvoch jedincoch je závislosť lineárna (tak ako pri selekcii ohodnotením). Viac ako troch jedincov v turnaji zvyčajne nepoužívame, lebo je príliš malá šanca, že sa vyberie nejaký jedinec zo slabšej polovice generácie.

Mutácia programu môže znamenať nielen zámenu inštrukcie alebo jej parametra na nejakom mieste, ale aj pridanie alebo ubranie inštrukcie, prípadne výmenu poradia inštrukcií.

Ďalšie informácie

Na základe často kladených otázok (faq) boli pridané nasledovné informácie:

Je vhodné vytvoriť jedincov len ako sekvencie pamäťových buniek a okrem nich jeden virtuálny stroj. Do tohto stroja sa **nakopíruje** príslušný jedinec a stroj začne vykonávať zodpovedajúci

program. Program je samomodifikujúci, takže pôvodná informácia ostáva zachovaná len v bunkách jedinca. Stroj na základe vygenerovaného výstupu (počtu nájdených pokladov, prípadne počtu vykonaných krokov) vygeneruje fitness pre zodpovedajúceho jedinca. Keď vytvorí fitness pre všetkých jedincov, tak sa prejde na vytvorenie novej generácie (ak ešte treba).

Pracujte vždy s pôvodnými jedincami! (Nie zmenenými vykonávaním.) Inak Vám evolúcia nebude konvergovať!

Hlavnou zložkou fitnes je počet nájdených pokladov. Ak chcete zakomponovať aj počet krokov (či už programu alebo panáčika, ktorý hľadá poklady), tak začnite od hodnoty jedna (aby bolo vždy od čoho odpočítavať) a za každý krok odpočítajte napríklad jednu tisícinu. Potom budú mať kratšie programy (riešenia) lepšiu fitnes ako dlhé, ale stále bude mať prednosť väčší počet nájdených pokladov.

Inštrukcie inkrementácie a dekrementácie sú cyklické, to znamená, že ak inkrementujem bunku so samými jednotkami, dostanem samé nuly a ak dekrementujem bunku so samými nulami, dostanem samé jednotky. Inkrementácia aj dekrementácia sa vykonáva nad celou bunkou, takže nemení len adresnú časť (posledných šesť bitov), ale v zodpovedajúcom prípade (111111 alebo 000000 na posledných šiestich miestach) aj typ inštrukcie.

Je možné mať viac typov mutácií. Napríklad invertovanie jedného bitu vo zvolenej bunke, naplnenie zvolenej bunky náhodným obsahom, invertovanie všetkých bitov zvolenej bunky, výmena obsahu dvoch susedných buniek a podobne. Čím ťažšia mutácia, tým menšia pravdepodobnosť jej výskytu by mala byť.

Odporúčané vstupné parametre programu: počet jedincov programu, typ selekcie, (prípadne typ kríženia) pravdepodobnosť mutácie (každého použitého typu), elitarizmus (áno/nie, prípadne počet), počet generácií na koniec/prerušenie, ak nenájdem všetky poklady skôr. Stačí pracovať s rozložením pokladov zo zadania, ale je možné si ich rozloženie načítavať zo súboru.

Dokumentácia

Dokumentácia musí obsahovať konkrétne použitý algoritmus (nie len náčrt algoritmu, ako v zadaní), podrobný opis všetkých použitých génov, inicializáciu prvej generácie a presný spôsob tvorby novej generácie. Dôležitou časťou dokumentácie je zhodnotenie vlastností vytvoreného systému a porovnanie dosahovaných výsledkov aspoň pre dva rôzne spôsoby tvorby novej generácie alebo rôzne spôsoby selekcie. Dosiahnuté výsledky (napr. vývoj fitness) je vhodné zobraziť grafom. Dokumentácia by mala tiež obsahovať opis vylepšovania, dolaďovania riešenia.
