

# Hľadanie pokladu

## Zadanie č. 2b

### Úloha

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



### Zadanie

Horeuvedenú úlohu riešite prostredníctvom evolučného programovania nad virtuálnym strojom.

Tento špecifický spôsob evolučného programovania využíva spoločnú pamäť pre údaje a inštrukcie. Pamäť je na začiatku vyplnená od prvej bunky inštrukciami. Za programom alebo od určeného miesta sú uložené inicializačné údaje (ak sú nejaké). Po inicializácii sa začne vykonávať program od prvej pamäťovej bunky. (Prvou je samozrejme bunka s adresou 000000.) Inštrukcie v pamäťových bunkách, môžu realizovať vetvenie, programové skoky, čítať nejaké údaje zo vstupu a prípadne aj zapisovať na výstup. Program končí inštrukciou na zastavenie, po stanovenom počte krokov, pri chybnnej inštrukcii, po úplnom alebo nesprávnom výstupe. Keď program sa ohodnotí na základe vyprodukovaného výstupu alebo, keď program nezapisuje na výstup, podľa výsledného stavu 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 (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 inštrukcie by mali byť jasné, pri poslednej je potrebné si dedefinovať, čo sa vypíše pri akej hodnote bunky. Napríklad ak bude 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 pamäti to bude **L**. Ako ukážku si uvedieme jednoduchý príklad:

<b>Adresa:</b>	000000	000001	000010	000011	000100	000101	000110	...
<b>Hodnota:</b>	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, vypísané hodnoty nezodpovedajú hodnotám na začiatku, ale počas behu programu.) Je to lepšie vidno na [jednoduchšej simulácii](#) (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. Musí sa dodržať podmienka maximálneho počtu pamäťových buniek 64 a limit 500 krokov programu. Rozšírenie inštrukcií sa nielen na zadenovanie nových typov inštrukcií, ale hlavne na vytvorenie inštrukcií s podmieneným vykonávaním.

### 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 bitov každej bunky 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. Vytvoril požadovaný počet populácií, algoritmus vypíše doteraz nájdené najlepšie riešenie a čaká na rozhodnutie používateľa – keď sa rozhodne, že chce ďalšie opakovanie.
5. Jednou z metód selekcie (ruleta, turnaj a iné) sa určia rodičia a krížením vytvorí 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 postupať je lepšia.

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

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

Ďalšie informácie, ktoré sa vám naozaj zídu

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 zostáva zachovaná len v bunkách jedinca. Stroj na základe vygenerovaného výstupu (počtu nájdených pokladov, prípadne počtu 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 fitness je počet nájdených pokladov. Ak chcete zakomponovať aj počet krokov (či už programu alebo panáčika hľadá poklady), tak začnite od hodnoty jedna (aby bolo vždy od čoho odpočítavať) a za každý krok odpočítajte napríklad jednu jednotku. Potom budú mať kratšie programy (riešenia) lepšiu fitness 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 jednotky a ak dekrementujem bunku so samými nulami, dostanem samé jednotky. Inkrementácia aj dekrementácia sa vykonáva nad celou bunkou, takže nemeňte 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 náhodnými bitmi, 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), elitizmus (áno/nie, prípadne počet), počet generácií na koniec/prerušenie, ak nenájdem všetky poklady. 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 algoritmu, porovnanie dosahovaných výsledkov aspoň pre dva rôzne spôsoby tvorby novej generácie alebo rôzne spôsoby selekcie. Dosiahnuté výsledky (vývoj fitness) je vhodné zobrazovať grafom. Dokumentácia by mala tiež obsahovať opis vylepšovania, doladovania riešenia.

---