Zápočtový program – Neprocedurální programování 2015/16

Rešič Pentomino

Marek Židek

Zadání:

Daná plocha se má přesně pokrýt 12 obrazci z 5 spojených čtverců Pentomino.

Řešení:

Vybral jsem 4 plochy z odkazu na wikipedii pokrytí obdélníku a původní koncept byl vypsat počet možných řešení pro vybranou plochu. Později jsem pochopil, že to bych se nedočkal, ale strukturu programu jsem takovou nechal, protože jsem na ní pěkně pochopil fungování findall/3. Program tedy hledá všechna řešení.

Bez heuristik:

Základ je takový, že se vezme určité pentomino a snaží se ve všech svých přeloženích a otočeních položit na každé možné políčko a dále rekurzivně zkoušet další obrazec pentomino.

Použité heuristiky:

používají se po uspěšném vložení obrazce a jsou true/false.

Heuristika1 a heuristika2 – hledá uzavřené volné plošky do hloubky 1 resp. 2 a je v podstatě natvrdo napsaná, aby se omezila složitost samotné heuristiky3.

Heuristika3 – hledá pro každý bod z něj souvislý dosažitelný graf a potřebuje, aby měl počet vrcholů dělitelný 5.

Dále mě napadla **heuristika4** – jako prvni 2 vkladat obrazce 'I' a po něm 'L' a dále vyhledávat linie po třech bodech ohranicenych po stranach, protoze žadné jiné obrazce nemají tvar pro zaplnění tohoto prostoru, ale po zkoumáni jednotlivých pokusů o řesení se toto nevyskytuje až tak často.(ale vyhled. strom by to jistě výhodně ořezalo).

Struktury:

Plocha reprezentována Listem listů jakožto řádků s celkovym počtem bodů na konci lehce přístupné last/2.

Obrazce jako predikáty a k nim predikáty přidělující všechny možné varianty obrazce (otočení, zrdcadlové převrácení) v podobě listu [1, 0, 2, 0, 3, 0, 4, 0] např. pro 'I' jakožto relativnich souřadnic od leveho horniho rohu.

Zajímavé (pro mě) **triky:**

- u metody **solve** nastane konec rekurze po nastavení akumulátoru na 12. Abych zachytil tento moment jako count pro všechna řešení, nastavím hodnotu Vysledek na 1. U findall/3 sleduji tuto proměnnou Vysledek v prvním argumentu a háži ji do Bagu ve třetím, dále jen spočtu velikost Bagu a mám požadovaný počet řešení.

- dále implementace **BFS** pro takovéto plochy mi vyšla překvapivě krátká:

```
%Kolik -> znaci pocet bodu souvisle nezaplnene plochy -> pote se testuje delitelnost 5ti
%Posledni argument znaci, jestli '1' nalezli jsme novy volny bod ve Fronte, nebo  '0' hledame pro vsechny sousedy a davame do fronty.
bFS(_,_,[],Kolik,Kolik,1).
bFS(Sirka, List, [H|T], Acc, Kolik, 1) :- volneMistoAZaber(List, H, NovaPlocha) ->
                    Acc1 is Acc + 1, bFS(Sirka, NovaPlocha, [H|T], Acc1, Kolik, 0);
                    bFS(Sirka, List, T, Acc, Kolik, 1).

bFS(Sirka, List, [H|T], Acc, Kolik, 0) :- NPoz1 is H + 1, NPoz2 is H - 1, NPoz3 is H + Sirka, NPoz4 is H - Sirka, bFS(Sirka, List, [Poz1,Poz2,Poz3, Poz4|T], Acc, Kolik, 1).
```

-zajímavé jsou dále ještě implementace heuristiky1,2 a 3.

Testovací data:

Nejlépe lze fungování sledovat v guitraceru, kde lze ověřit korektní fungování ořezávání nesplnitelných větví pomocí heuristik a spypointu na nich.
Jinak je nyní program napsán tak, že místo '1' jako zaplnění plochy vkládá písmena označení obrazců podle wikipedie a po spuštění nezávisle na výběru plochy předvede **testovací vstup** i s grafickým výstupem. Po vytisknutí program počítá všechna řešení.

I přes použité heuristiky, někdy jedno řešení trvá velmi dlouho a proto je u testovacího vstupu nastavena orientace požadovaného řešení prvních 3 z 12 obrazců jako hned první možnost v backtrackingu, aby se urychlil výstup.

Závěr:

Bohužel s vervou si něco naprogramovat v prologu, mě napadlo zapátrat po internetu po nějakém teoretickém řešení a naimplementovat jej až po týdnu psaní programu a konec zkouškového se blížíl. Odevzdávám tedy prohledávání obohacené o popsané heuristiky, avšak z implementace jsem si mnoho odnesl.