Mateusz Koślacz Warszawa, 15.01.2015 r.

Andrzej Słoma

**Spop – Opis projektu**

Plik *txt* przygotowany do wczytania w programie powinien mieć format:

* pierwszy wiersz jest listą liczb z lewej strony planszy,
* drugi wiersz jest listą liczb nad planszą,
* trzeci wiersz jest listą par opisujących położenie domków w postaci (wiersz, kolumna) kolejność par na liście może być dowolna.

|  |  |
| --- | --- |
| **Przykładowa zawartość pliku wejściowego** | **Format rozwiązania wyświetlany w konsoli i zapisywany do pliku** |
| [1, 0, 2, 1, 2, 1]  [1, 1, 2, 1, 1, 1]  [(0, 1), (3, 2), (3, 4), (4, 0), (4, 4), (5, 2), (5, 5)] | 1 1 2 1 1 1  1 \* h t \* \* \*  0 \* \* \* \* \* \*  2 \* \* t \* t \*  1 t \* h \* h \*  2 h \* \* t h t  1 \* t h \* \* h  Legenda:  h - domek  t - zbiornik  \* - pole puste |

Program uruchamiamy za pomocą funkcji ***main*** umieszczonej w module ***Main.hs***. Po uruchomieniu programu należy z menu:

r NazwaPlikuWej - wczytaj plik NazwaPlikuWej, wyswietl rozwiazanie w konsoli

w NazwaPlikuWej NazwaPlikuWyj - wczytaj plik NazwaPlikuWej, zapisz rozwiazanie do pliku NazwaPlikuWyj

e - koniec

wybrać (wpisać) odpowiednie polecenie. Sprawdzana jest poprawność wpisanego polecenia (podane ścieżki plików, istnienie pliku wejściowego) oraz poprawność formatu pliku wejściowego. Przykładowe poprawne i błędne polecenia wczytania pliku txt:

|  |  |
| --- | --- |
| **Plecenie** | **Błąd** |
| r test\_files/p1.txt | brak |
| r | \*\*\* Exception: Nie podano nazwy pliku wejsciowgo |
| r test\_files/aaa.txt | \*\*\* Exception: Nie istnieje plik: test\_files/aaa.txt |
| r test\_files/zly.txt | \*\*\* Exception: Zly format pliku wejsciowego |
| w test\_files/p1.txt outfile.txt | brak |
| w test\_files/p1.txt | \*\*\* Exception: Nie podano nazwy pliku wyjsciowego |

Dodatkowo przygotowaliśmy moduł testowy *Test.hs*, folder z przykładowymi plikami wejściowymi *test\_files* oraz rozwiązaniami *sol\_files*. Uruchomienie wszystkich plików testowych odbywa się za pomocą funkcji *solverTestAll* (z *Test.hs*)

Algorytm rozwiązujący zagadki działa eliminując pola, które w danym kroku zostały ocenione jako takie, w których na pewno nie może znajdować się zbiornik (przeanalizowane), a następnie, gdy w okolicach, w których musi się znajdować zbiornik pozostaje tylko jedno wolne (nieprzeanalizowane) miejsce, wstawia tam zbiorniki, aktualizuje sąsiedztwo pól i powtarza te czynności, aż na planszy nie zostanie ani jedno wolne miejsce. Algorytm przeprowadza analizę wykonując następujące kroki:

* Sprawdzenie kolumn i wierszy planszy na podstawie liczb przy brzegach:
  + liczba zbiorników w danej linii jest równa liczbie przy brzegu – zamienia wolne miejsca w linii na przeanalizowane,
  + liczba wolnych miejsc jest mniejsza niż liczba przy brzegu pomniejszona o liczbę już wstawionych zbiorników w linii – zamienia wolne miejsca na zbiorniki.
* Eliminacja nieprzeanalizowanych pól, w których na pewno nie może znajdować się zbiornik, ze względu na sąsiedztwo innego zbiornika, lub brak sąsiedztwa domku.
* Wstawienie zbiorników w sąsiedztwie domków bez sąsiadującego zbiornika, wokół których zostało tylko jedno nieprzeanalizowane pole.
* Eliminacja nieprzeanalizowanych pól wokół domków, które mają podłączony zbiornik. Analiza podłączenia zbiornika :
  + domek ma sąsiadujący zbiornik, i czy ten zbiornik sąsiaduje tylko z tym domkiem – podlączony,
  + z domkiem sąsiadują dwa lub więcej zbiorników – podłączony,
  + pozostałe sytuacje – niepodłączony.

Gdy algorytm napotka sytuację, w której uzna, że wstawienie zbiornika w każdym z pozostałych miejsc będzie równoważne, wstawia go w pierwszym napotkanym wolnym miejscu, sprawdzająć od lewej do prawej, od góry do dołu.

Przedstawiona procedura pozwoliła osiągnąć dobre wyniki w 100% testów przeprowadzonych na 10 przykładowych planszach ze strony <http://www.wydawnictwologi.pl/lamiglowki-architekta.html> .

Dokładny sposób implementacji jest opisany w komentarzach w kodzie. Algorytm rozwiązujący zagadki znajduje się w pliku Solver.hs.