Haskell: Roll the Ball

- Responsabili:
  - Viorica Mancaș [mailto:vioricamancas@yahoo.com]
  - Ionuț Bălășescu [mailto:imbalasescu@gmail.com]
  - Andrei Medar [mailto:andreimedar@gmail.com]
  - Gabriel Dănuț Matei [mailto:matei.danut.dm@gmail.com]
  - Vlad Neculae [mailto:neculae.vlad@gmail.com]
  - Mihnea Muraru [mailto:mmihnea@gmail.com]

- Deadline soft: **03.05.2020**
- Deadline hard: 03.05.2020
- Data publicării: 10.04.2020
- Data ultimei modificări: 30.04.2020 changelog
- Tema se va încărca pe **vmchecker [https://vmchecker.cs.pub.ro/ui/#PP]**
- Data checker-ului: 10.04.2020
- Forum temă [https://acs.curs.pub.ro/2019/mod/forum/view.php?f=2508]

Obiective

- Utilizarea mecanismelor **funcționale**, de **tipuri** și de **evaluare leneșă** din limbajul Haskell pentru rezolvarea unei probleme de **căutare** în spațiul stărilor.
- Exploatarea evaluării leneșe pentru **decuplarea conceptuală** a etapelor de construcție și de explorare a acestui spațiu.

Dependențe

Pentru a rula checker-ul, este necasară instalarea unor dependențe adiționale. Putem face asta urmând următorii pași:

1. Adăugăm psqueues la dependențele lui stack, modificând linia extra-deps din fișierul stack.yaml astfel: `extra-deps: [random-1.1, psqueues-0.2.7.2]`
2. Rulăm `stack install unordered-containers` si `stack install astar` pentru a instala pachetele necesare.

Descriere

Tema urmărește implementarea unei variante simplificate a jocului ***Roll the Ball [http://fun.touchpal.com/game/fourj-roll-the-ball-online/index.html]***, și a unui mecanism de rezolvare a oricărui nivel, utilizând **căutare leneșă** în spațiul stărilor. În acest sens, se va întrebuința o ***căutare bidirecțională [https://www.geeksforgeeks.org/bidirectional-search/]***, bazată pe **două parcurgeri în lățime**.

Roll the Ball

Jocul presupune deplasarea unor **celule** care conțin culoare de trecere de diverse forme și orientări, astfel încât **bila** să poată urma **traseul** de la poziția inițială (în imaginea următoare, în stânga-sus) la poziția finală (dreapta-jos).

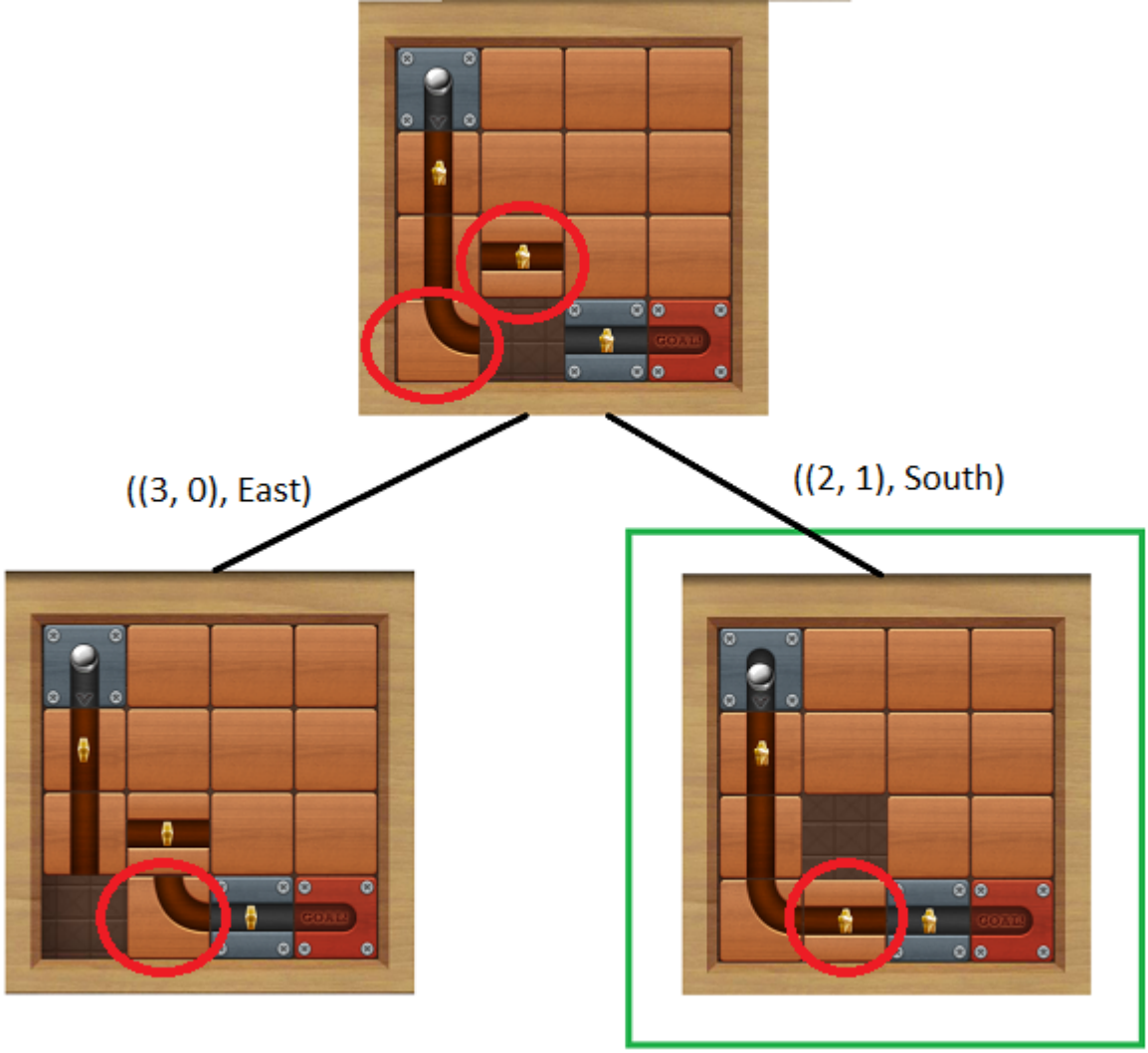
Putem observa următoarele elemente:

1. Celula de ***Start***: Aceasta va conține inițial bila și se poate afla oriunde pe tabla de joc. De asemenea va avea o direcție în care bila se va putea deplasa. Este o celulă **fixată**.
2. Celula de ***Win***: Marcată cu roșu, va fi poziția la care va trebui să ajungă bila în final. La fel ca celula de start, aceasta nu se poate muta și se poate afla oriunde pe tablă.
3. Celule care se pot muta: Pot avea sau nu **culoar**. Acestea trebuie să formeze un traseu continuu.
4. Celulele fixe: În implementarea jocului, pentru a simplifica reprezentarea, vom considera celulele fixe ca fiind celule de tip culoar care se pot muta.

Va încurajăm să încercați jocul [http://fun.touchpal.com/game/fourj-roll-the-ball-online/index.html] înainte de implementarea temei 😀.

Spațiul stărilor problemei

Căutarea soluției se va desfășura în **spațiul stărilor** problemei. Acesta este reprezentat de un graf, în care nodurile sunt **configurațiile** tablei de joc, iar muchiile, **acțiunile** care asigură tranzițiile între stări. Mai jos, este prezentat spațiul stărilor pentru nivelul de mai sus. O acțiune este reprezentată printr-o pereche (poziție, direcție), reprezentând în ce „direcție“ s-a deplasat celula de la poziția „poziție“. O poziție este reprezentată astfel: (linie, coloană), începând cu (0, 0) - colțul din stânga-sus.



Cu verde, este prezentată starea finală, care asigură rezolvarea nivelului.

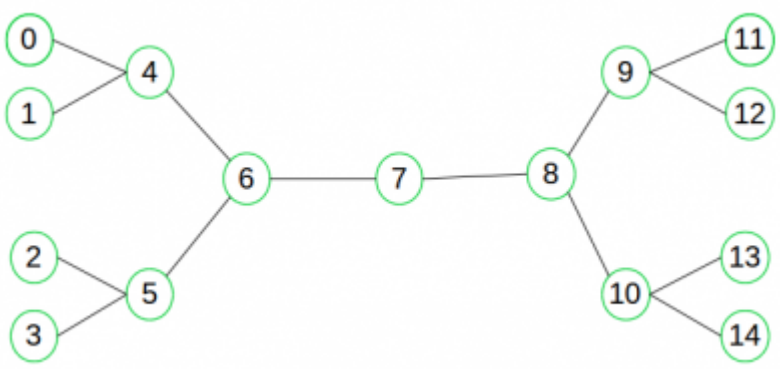
Având această reprezentare, vom putea enumera stările utilizând **parcurgerea în lățime** (deși ar putea fi utilizată în alte situații și parcurgerea în adâncime, de exemplu).

Atenție! Acțiunile sunt **reversibile**, ceea ce înseamnă că, din starea din stânga-jos, s-ar putea reveni în starea inițială prin acțiunea `( (3, 1), West )`. Acest lucru înseamnă că, la realizarea parcurgerii, este necesară reținerea stărilor **vizitate**, pentru **evitarea** ciclurilor.

Căutare bidirecțională

Strategia de căutare pe care o vom utiliza în temă este **bidirecțională**, bazată pe **două parcurgeri în lățime**. Astfel, atât starea inițială, cât și starea finală, se consideră cunoscute. Pe baza acestora, se realizează câte o parcurgere în lățime pornind din fiecare dintre cele două stări. Ele se desfășoară **în paralel**, expandând **alternativ** câte un nod din fiecare, până în momentul în care frontierele lor **se intersectează**.

Mai jos, este prezentat un **exemplu** de desfășurare a parcurgerii bidirecționale în lățime, pe baza următorului graf [https://www.geeksforgeeks.org/bidirectional-search/], unde nodul inițial este 0, iar cel final, 14.



În fiecare pas, este prezentată **frontiera** fiecăreia dintre cele două parcurgeri. Nodul ~~tăiat~~ este scos din frontieră, **vecinii** lui fiind adăugați în loc.

Pas	Frontiera 1	Frontiera 2
0	0	14
1	~~0~~ **4**	~~14~~ **10**
2	~~4~~ **1 6**	~~10~~ **13 8**
3	~~1~~ 6	~~13~~ 8
4	~~6~~ **5** 7	~~8~~ **9** 7

Se observă că, în pasul 4, după generarea nodului 7 de către a doua parcurgere, acesta este **găsit** în frontiera primei parcurgeri, și căutarea se oprește.

Dacă frontiera devine goală, funcția se va termina.

Cerințe

Rezolvarea temei este structurată pe etapele de mai jos. Începeți prin a vă familiariza cu structura **arhivei** de resurse. Va fi de ajutor să parcurgeți indicațiile din enunț **în paralel** cu comentariile din surse. În rezolvare, exploatați testele drept **cazuri de utilizare** a funcțiilor pe care le implementați.

Implementarea jocului și afișarea (35p)

Elementele care compun jocul, **mecanica** și **afișarea** jocului se vor realiza în fișierul `RollTheBall.hs`. Va trebui să completați propriile definiții pentru tipurile de date din fișier, urmărind TODO-urile.

Pentru reprezentarea unui nivel recomandăm folosirea modulului `Data.Array [http://hackage.haskell.org/package/array-0.5.3.0/docs/Data-Array.html]`. Recomandăm să urmăriți de asemenea și testele pentru a va asigura că implementarea voastră este corectă.

Detalii despre folosirea modulului `Data.Array` puteți găsi și în exemplul din secțiunea Resurse.

Rezolvarea **afișării nivelului** se va realiza prin implementarea instanței de `Show` atât pentru **nivel** cât și pentru obiectele care îl vor compune, care vor avea tipul `Cell`.

Observație: Pentru afișare, trebuie să includeți un caracter '\n' la începutul nivelului și un '\n' după fiecare linie din tabla de joc.

Vizualizare

Având implementate toate cele de mai sus, veți putea juca în linia de comandă. Pentru a realiza acest lucru rulați fișierul `Interactive.hs` și apelați funcția `play`, pasând level-ul ca argument. De exemplu, `play level3`. Toate level-urile sunt predefinite în fișierul `RollLevels.hs`.

Înainte de rulare setați valoarea `workingOs` la `Windows` sau `Linux` în funcție de sistemul pe care rezolvați tema.

Rezolvarea jocului (65p)

În continuare, pentru a ajunge la starea finală va trebui să reprezentăm spațiul stărilor și să îl parcurgem. În fișierul `ProblemState.hs` veți găsi clasa care va interfața în mod generic funcțiile pentru generarea spațiului stărilor. În fișierul `RollTheBall.hs` veți crea o instanța a clasei **`ProblemState`** pentru jocul din enunț cu tipurile `Level` și `(Position,Directions)`.

Apoi, în fișierul `Search.hs` va trebui să va construiți tipul de date pentru a reprezenta arborele stărilor și să implementați funcția care va genera „tot“ spațiul (**`createStateSpace`**).

Având spațiul generat vom putea implementa funcțiile `bidirBFS` și `bfs`. Funcția **`bfs`** va primi un nod de pornire și va întoarce un **flux** de perechi formate din lista nodurilor ce au fost adăugate în frontieră la un anumit pas, respectiv frontiera (lista de noduri). Funcția **`bidirBFS`** va primi cele două noduri (inițial și final) și, folosind fluxurile generate de `bfs`, va întoarce o pereche de două noduri ce conțin aceeași stare (cele două noduri făcând parte din cele două parcurgeri).

Motivul pentru care primul element al perechilor întoarse de funcția `bfs` reprezintă lista nodurilor ce au fost adăugate în frontieră la acel pas este că astfel putem să verificăm apartenența la cealaltă frontieră **doar** pentru acele noduri. Astfel, **nu** suntem nevoiți ca la fiecare pas să verificăm dacă fiecare element dintr-o frontieră se regăsește și în cealaltă.

De asemenea, această implementare folosind fluxuri ne permite să observăm **evoluția frontierei în timp**, spre deosebire de o abordare imperativă ce presupunea modificarea ei *in place*.

Având posibilitatea de a găsi intersecția dintre cele două frontiere, vom dori să generăm și **calea** de la starea inițială la cea finală. Pentru aceasta va trebui să implementați funcția **`extractPath`**.

Consultați fișierul `README` din arhiva de testare pentru a vedea cum se rulează atât checkerul, cât și `play`.

Bonus: Euristică (20p)

Dorim să **accelerăm** găsirea soluției, vizitând mai întâi noduri cu o probabilitate mai mare de a ne conduce spre starea finală. În acest sens, vom folosi algoritmul **A* [https://en.wikipedia.org/wiki/A*_search_algorithm]**, implementat în modulul `Data.Graph.AStar` `[http://hackage.haskell.org/package/astar-0.3.0.0/docs/Data-Graph-AStar.html]`, pe care îl puteți instala cu comanda `stack install astar`. Funcția de interes din modul este **`aStar` `[http://hackage.haskell.org/package/astar-0.3.0.0/docs/Data-Graph-AStar.html#v:aStar]`**, pentru care va trebui să definiți parametrii în fișierul `AStarHeuristic.hs`.

Majoritatea parametrilor lui `aStar` se obțin **traducând** în forma solicitată funcțiile implementate deja de voi în celelalte părți alte temei. În plus, va trebui să implementați de la zero **euristica** specifică problemei noastre, în funcția **`nonTrivialHeuristic`**. Testele vor compara numerele de **stări expandate intern** de funcția `aStar` (nu lungimea soluției întoarse) în scenariul euristicii voastre și în cel al unei euristici banale, constante. Se așteaptă ca numărul de stări expandate prin euristica voastră să fie **mai mic**.

De asemenea, veți observa că primul parametru al funcției `aStar` este o funcție care generează **vecinii** unui nod sub forma unui `HashSet` `[https://hackage.haskell.org/package/unordered-containers-0.2.10.0/docs/Data-HashSet.html#t:HashSet]`. Pentru aceasta, este necesar să instanțiați clasa **`Hashable` `[https://hackage.haskell.org/package/hashable-1.2.7.0/docs/Data-Hashable.html#t:Hashable]`** cu tipurile `Level` și `Cell`, conform indicațiilor din fișierul `AStarHeuristic.hs`.

Precizări

1. **Atenție!** Funcțiile `show`, `createLevel`, `successors`, `bidirBfs` și `extractPath` trebuie implementate **fără** recursivitate explicită. Nerespectarea acestei cerințe va conduce la **penalizări** de 2p din 100 per funcție.
2. Exploatați cu încredere ***pattern matching*** și **gărzi**, în locul *if*-urilor imbricate.
3. Având în vedere că unele funcții din modulele `Data.Array` și `Data.Set` au același nume cu funcțiile pe liste (exemple: `map`, `filter`), **conflictele de nume** se rezolvă prin următorul mecanism:
   - Importarea modulelor se face printr-un **alias**: `import qualified Data.Set as S`.
   - Tipurile și funcțiile din acel modul se menționează întotdeauna **prefixate** de acel alias: `S.Set`, `S.insert` etc.
4. Pentru rularea **testelor**, executați comanda `stack runhaskell RollTest`.
5. Arhiva pentru **vmchecker** este suficient să conțină fișierele `RollTheBall.hs`, `Search.hs` și `AStarHeuristic.hs`.
6. Din cauza resurselor limitate, varianta de checker de pe vmchecker nu va include testele aferente `level5`.

Resurse

- **Schelet și checker**
- **Exemplu Array**

Changelog

02.05.2020 Actualizat scheletul cu tipul corect al funcției `connection` (conform 15.04), care se modificase din greșeală după schimbarea din 30.04.
30.04.2020 Eliminat testul pe level1 de la bonus, redistribuit punctajele pe restul testelor.
28.04.2020 Modificarea checkerului pentru ca functia BFS sa nu fie afecatata negativ de verificarea vecinilor deja vizitati.
19.04.2020 Clarificare condiție de oprire a funcției de căutare.
18.04.2020 Clarificare instanță ProblemState și rezolvare diferențe enunț-schelet.
16.04.2020 Adăugat informații adiționale despre instalatul dependențelor necesare checker-ului.
15.04.2020 Modificarea semnăturii funcției connection din fișierul `RollTheBall.hs` din forma `connection :: Cell → Cell → Bool` in `connection :: Cell → Cell → Directions → Bool`.
14.04.2020 Precizare privind afișarea nivelului.