ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

Τμήμα Πληροφορικής

Προαιρετική εργασία για το μάθημα «Τεχνητή Νοημοσύνη και Έμπειρα Συστήματα»

ΚΩΝΣΤΑΝΤΙΝΟΣ-ΖΩΗΣ ΣΥΡΙΟΣ – Π16139

ΗΜ/ΝΙΑ ΠΑΡΑΔΟΣΗΣ – 30/05/2019

1. Εκφώνηση Εργασίας

Αναπτύξτε πρόγραμμα επίλυσης του προβλήματος των ιεραποστόλων και κανιβάλων με χρήση ενός τυφλού και ενός ευρετικού αλγορίθμου της επιλογής σας και σε γλώσσα προγραμματισμού της επιλογής σας.

Παραδοτέα της εργασίας είναι μία σύντομη αναφορά που να περιλαμβάνει τον τρόπο δράσης του υπολογιστή σύμφωνα με τον αλγόριθμο επίλυσης και παραδείγματα εκτέλεσης του προγράμματος που αναπτύξατε.

2. Γλώσσα προγραμματισμού και περιβάλλον ανάπτυξης

Η γλώσσα που χρησιμοποιήθηκε για την εκπόνηση της εργασίας είναι η αντικειμενοστρεφής C#. Πρόκειται για μία εφαρμογή κονσόλας που αναπτύχθηκε στο Visual Studio 2017.

3. Αλγόριθμοι και λύση

Για τυφλό αλγόριθμο επέλεξα τον Depth First ενώ για τον ευρετικό τον Best-First.

3.1. Κοινές Κλάσεις

3.1.1. RiverBank

Η κλάση RiverBank αναπαριστά μία όχθη του ποταμού και περιέχει τον αριθμό των κανίβαλων και των ιεραπόστολων, καθώς και συνάρτηση για να ελέγχει αν τα στοιχεία της όχθης είναι έγκυρα για το πρόβλημα (π.χ. λιγότεροι ή ίσοι κανίβαλοι από ότι ιεραπόστολοι)

3.1.2. State

Η κλάση State αναπαριστά στιγμιότυπα του προβλήματος. Περιλαμβάνει δύο αντικείμενα RiverBank για την αριστερή και δεξιά όχθη και μία μεταβλητή ώστε να σημειώνεται η θέση της βάρκας. Επίσης, γίνεται χρήση ενός static μετρητή για να δίνεται ένα id σε κάθε καινούριο State. Υπάρχουν δύο τύποι constructor: ένας για την κατασκευή της αρχικής και τελικής κατάστασης και ένας για τις ενδιάμεσες. Τέλος, η κλάση State περιέχει συναρτήσεις οι οποίες ελέγχουν την εγκυρότητα της κατάστασης, το αν μία κατάσταση είναι ίδια με κάποια άλλη (ίσος αριθμός κανίβαλων και ιεραπόστολων σε κάθε όχθη με την βάρκα στο ίδιο σημείο) και συνάρτηση για την δημιουργία των καταστάσεων παιδιών σύμφωνα με τους πέντε τελεστές του προβλήματος (κκ, ικ, ιι, ι, κ).

3.1.3. ClosedSet

Η ClosedSet αναπαριστά το κλειστό σύνολο καταστάσεων, δηλαδή όσων έχουν ήδη εξεταστεί από τον αλγόριθμο. Περιλαμβάνει συναρτήσεις για την σύγκριση των καταστάσεων με μία καινούρια ώστε να μην ελεγχθεί χωρίς λόγο. Υλοποιείται ως λίστα από States.

3.1.4. Log

Η log περιλαμβάνει συναρτήσεις για την καταγραφή της πορείας του αλγόριθμου σε txt αρχείο, το οποίο βρίσκεται στον ίδιο φάκελο με το εκτελέσιμο.

3.2. Depth-first Search

3.2.1. Frontier

Η κλάση Frontier αναπαριστά το μέτωπο αναζήτησης του αλγόριθμου. Για τις απαιτήσεις του DFS την έχω υλοποιήσει ως μία στοίβα.

3.2.2. Εκτέλεση Προγράμματος

Αρχικά, το πρόγραμμα κατασκευάζει το αντικείμενο log για την παρακολούθηση του αλγόριθμου και στη συνέχεια κατασκευάζει την αρχική και την τελική κατάσταση, αρχικοποιεί το μέτωπο αναζήτησης και το κλειστό σύνολο καταστάσεων, φτιάχνει την τωρινή κατάσταση ως null και εισάγει την αρχική κατάσταση στο μέτωπο αναζήτησης.

```
Log log = new Log();

State InitialState = new State("initial");
log.WriteLine("Initial State created...\n");

State GoalState = new State("goal");
log.WriteLine("Goal State created...\n\n");


Frontier frontier = new Frontier();
ClosedSet closedSet = new ClosedSet();

State currentState = null;

frontier.Add(InitialState);
```

Έπειτα, το πρόγραμμα εισέρχεται στην επανάληψη η οποία τερματίζει όταν το μέτωπο αναζήτησης είναι άδειο.
Στην επανάληψη αυτή η πρώτη κατάσταση από το μέτωπο αναζήτησης γίνεται τωρινή.

```
currentState = frontier.Remove();
```

Ύστερα γίνεται έλεγχος για το αν η τωρινή κατάσταση βρίσκεται στο κλειστό σύνολο ή είναι τελική. Στην πρώτη περίπτωση, το πρόγραμμα ξαναπηγαίνει στην αρχή της επανάληψης ενώ βγαίνει από αυτήν στην δεύτερη περίπτωση.

```
if (closedSet.contains(currentState))
{
    // Log '-' for the children list if the closed set contains current state
    log.WriteLine(" | - ");
    continue;
}
else if (currentState.equals(GoalState))
{
    log.Write(" | GOAL STATE");
    log.WriteLine("\nSolution found!");
    break;
}
```

Αν τίποτα από τα παραπάνω δε συμβαίνει τότε παράγονται τα παιδιά της τωρινής κατάστασης, αφού ελεγχθούν για το αν έχουν δημιουργηθεί ήδη, και προστίθενται στο μέτωπο αναζήτησης, ενώ η τωρινή στο κλειστό σύνολο.

```
foreach(State expandingState in currentState.expandState(closedSet))
{
    printChildren += expandingState.getID() + ", ";
    frontier.Add(expandingState);
}
log.WriteLine(printChildren.TrimEnd().TrimEnd(','));
closedSet.add(currentState);
```

Στο ενδιάμεσο παρεμβάλλονται συναρτήσεις του log για την σημείωση της πορείας του αλγόριθμου.

3.2.3. Παράδειγμα Εκτέλεσης

Step n : <Frontier> | {Closed Set} | Current State | Children

```
Step 1: <1> | {} | 1 | 2, 3, 4
Step 2: <4, 3, 2> | {1} | 4 | 1
Step 3: <1, 3, 2> | {1, 4} | 1 | -
Step 4: <3, 2> | {1, 4} | 3 | 1, 5
Step 5: <5, 1, 2> | {1, 4, 3} | 5 | 6, 7, 3
Step 6: <3, 7, 6, 1, 2> | {1, 4, 3, 5} | 3 | -
Step 7: <7, 6, 1, 2> | {1, 4, 3, 5} | 7 | 1, 5
Step 8: <5, 1, 6, 1, 2> | {1, 4, 3, 5, 7} | 5 | -
Step 9: <1, 6, 1, 2> | {1, 4, 3, 5, 7} | 1 | -
Step 10: <6, 1, 2> | {1, 4, 3, 5, 7} | 6 | 5, 8
Step 11: <8, 5, 1, 2> | {1, 4, 3, 5, 7, 6} | 8 | 9, 6
Step 12: <6, 9, 5, 1, 2> | {1, 4, 3, 5, 7, 6, 8} | 6 | -
Step 13: <9, 5, 1, 2> | {1, 4, 3, 5, 7, 6, 8} | 9 | 8, 10
Step 14: <10, 8, 5, 1, 2> | {1, 4, 3, 5, 7, 6, 8, 9} | 10 | 11, 9
Step 15: <9, 11, 8, 5, 1, 2> | {1, 4, 3, 5, 7, 6, 8, 9, 10} | 9 | -
Step 16: <11, 8, 5, 1, 2> | {1, 4, 3, 5, 7, 6, 8, 9, 10} | 11 | 10, 12
Step 17: <12, 10, 8, 5, 1, 2> | {1, 4, 3, 5, 7, 6, 8, 9, 10, 11} | 12 | 13, 11
Step 18: <11, 13, 10, 8, 5, 1, 2> | {1, 4, 3, 5, 7, 6, 8, 9, 10, 11, 12} | 11 | -
Step 19: <13, 10, 8, 5, 1, 2> | {1, 4, 3, 5, 7, 6, 8, 9, 10, 11, 12} | 13 | 12, 14, 15
Step 20: <15, 14, 12, 10, 8, 5, 1, 2> | {1, 4, 3, 5, 7, 6, 8, 9, 10, 11, 12, 13} | 15 | 16, 13
Step 21: <13, 16, 14, 12, 10, 8, 5, 1, 2> | {1, 4, 3, 5, 7, 6, 8, 9, 10, 11, 12, 13, 15} | 13 | -
Step 22: <16, 14, 12, 10, 8, 5, 1, 2> | {1, 4, 3, 5, 7, 6, 8, 9, 10, 11, 12, 13, 15} | 16 | GOAL STATE
Solution found!
```

3.3. Best-First Algorithm

3.3.1. Ευρετική Συνάρτηση

Για την ευρετική συνάρτηση χαλαρώνω το πρόβλημα αφαιρώντας τους εξής 2 περιορισμούς:

- Κανίβαλοι λιγότεροι ή ίσοι από ιεραπόστολους σε κάθε όχθη
- Η βάρκα χρειάζεται ένα άτομο για να επιστρέψει στην αριστερή όχθη

Από το παραπάνω χαλαρό πλέον πρόβλημα προκύπτει η ευρετική συνάρτηση:

h(n) = (#ΚανίβαλωνΑριστερά + #ΙεαρπόστολωνΑριστερά) / Χωρητικότητα-Βάρκας

3.3.2. Frontier

Η κλάση Frontier αναπαριστά το μέτωπο αναζήτησης του αλγόριθμου. Για τις απαιτήσεις του BestFS την έχω υλοποιήσει ως μία ταξινομημένη βάσει της τιμής της ευρετικής συνάρτησης λίστας που περιέχει States. Η τιμή της ευρετικής συνάρτησης υπολογίζεται κατά την κατασκευή του State.

3.3.3. Εκτέλεση Προγράμματος

Η εκτέλεση του BestFS είναι ίδια με αυτή του DFS αλγοριθμικά με την διαφορά ότι η επόμενη current state είναι πάντα η πρώτη του μετώπου αναζήτησης καθώς αυτό είναι διαρκώς ταξινομημένο βάσει της ευρετικής συνάρτησης (μικρότερο σε μεγαλύτερο).

3.3.4. Παράδειγμα Εκτέλεσης

Step n : <Frontier(HeuristicValue)> | {Closed Set} | Current State | Children

```
Initial State created...

Goal State created...


Step 1: <1(3)> | {} | 1 | 2, 3, 4
Step 2: <3(2), 2(2), 4(2.5)> | {1} | 3 | 1, 5
Step 3: <2(2), 5(2.5), 4(2.5), 1(3)> | {1, 3} | 2 | 1, 6
Step 4: <6(2.5), 5(2.5), 4(2.5), 1(3), 1(3)> | {1, 3, 2} | 6 | 7, 2, 3
Step 5: <7(1.5), 3(2), 2(2), 5(2.5), 4(2.5), 1(3), 1(3)> | {1, 3, 2, 6} | 7 | 6, 8
Step 6: <8(2), 3(2), 2(2), 6(2.5), 5(2.5), 4(2.5), 1(3), 1(3)> | {1, 3, 2, 6, 7} | 8 | 9, 7
Step 7: <9(1), 7(1.5), 3(2), 2(2), 6(2.5), 5(2.5), 4(2.5), 1(3), 1(3)> | {1, 3, 2, 6, 7, 8} | 9 | 8, 10
Step 8: <7(1.5), 10(2), 8(2), 3(2), 2(2), 6(2.5), 5(2.5), 4(2.5), 1(3), 1(3)> | {1, 3, 2, 6, 7, 8, 9} | 7 | -
Step 9: <10(2), 8(2), 3(2), 2(2), 6(2.5), 5(2.5), 4(2.5), 1(3), 1(3)> | {1, 3, 2, 6, 7, 8, 9} | 10 | 11, 9
Step 10: <9(1), 11(1), 8(2), 3(2), 2(2), 6(2.5), 5(2.5), 4(2.5), 1(3), 1(3)> | {1, 3, 2, 6, 7, 8, 9, 10} | 9 | -
Step 11: <11(1), 8(2), 3(2), 2(2), 6(2.5), 5(2.5), 4(2.5), 1(3), 1(3)> | {1, 3, 2, 6, 7, 8, 9, 10} | 11 | 10, 12
Step 12: <12(1.5), 10(2), 8(2), 3(2), 2(2), 6(2.5), 5(2.5), 4(2.5), 1(3), 1(3)> | {1, 3, 2, 6, 7, 8, 9, 10, 11} | 12 | 13, 11
Step 13: <13(0.5), 11(1), 10(2), 8(2), 3(2), 2(2), 6(2.5), 5(2.5), 4(2.5), 1(3), 1(3)> | {1, 3, 2, 6, 7, 8, 9, 10, 11, 12} | 13 | 12, 14, 15
Step 14: <15(1), 14(1), 11(1), 12(1.5), 10(2), 8(2), 3(2), 2(2), 6(2.5), 5(2.5), 4(2.5), 1(3), 1(3)> | {1, 3, 2, 6, 7, 8, 9, 10, 11, 12, 13} | 15 | 16, 13
Step 15: <16(0), 13(0.5), 14(1), 11(1), 12(1.5), 10(2), 8(2), 3(2), 2(2), 6(2.5), 5(2.5), 4(2.5), 1(3), 1(3)> | {1, 3, 2, 6, 7, 8, 9, 10, 11, 12, 13, 15} | 16 | GOAL STATE
Solution found!
```