ΑΝΑΦΟΡΑ (Α' ΜΕΡΟΣ)

Tom & Jerry : Κανίβαλοι(cats) και ιεραπόστολοι(mice)

Βανταράκη Αικατερίνη // 3210020

Γεωργακά Ευαγγελία // 3170023

Χασιώτη Ευαγγελία // 3120203

ΠΕΡΙΕΧΟΜΕΝΑ

2. Μοντελοποίηση του προβήματος

3. Εφαρμογή του A* για την αναζήτηση της Βέλτιστης Λύσης

4. Επεξήγηση κώδικα

****

1. Εισαγωγή

(i) **Μοντελοποίηση**: Χρησιμοποιούμε την κλάση State για να αναπαραστήσουμε τις καταστάσεις του συστήματος (θέση των γατών, ποντικιών και της βάρκας). Οι καταστάσεις συνδέονται μεταξύ τους μέσω της αναζήτησης A* για να βρούμε την καλύτερη λύση.

(ii) **Αναζήτηση**: Η αναζήτηση A* στην κλάση SpaceSearcher καθοδηγείται από τη συνάρτηση κόστους f(n), που υπολογίζεται από τις παραμέτρους g(n) και h(n).

(iii) **Επιδόσεις**: Ο αλγόριθμος A* βρίσκει την βέλτιστη πορεία μεταφοράς, αποφεύγοντας τα λάθη στις μετακινήσεις (π.χ. περισσότερες γάτες από ποντίκια σε οποιαδήποτε πλευρά).

Το πρόβλημα αυτό μοντελοποιείται ως ένα πρόβλημα αναζήτησης σε ένα γράφο, όπου κάθε κατάσταση αντιπροσωπεύει μια διάταξη των ζώων στις δύο πλευρές του ποταμού και τη βάρκα, και η αναζήτηση πρέπει να βρει την ακολουθία των μεταφορών (καταστάσεων) που οδηγούν στην επίλυση του προβλήματος.

2. Μοντελοποίηση του Προβλήματος

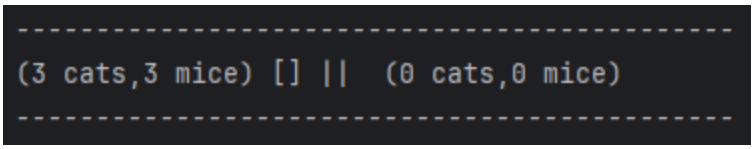
Χώρος Καταστάσεων

Ο **χώρος καταστάσεων** αναπαριστά όλες τις δυνατές καταστάσεις του προβλήματος. Κάθε κατάσταση αντιπροσωπεύει μια συγκεκριμένη διάταξη των γατών, των ποντικιών και της βάρκας στις δύο πλευρές του ποταμού.

Η κατάσταση καθορίζεται από τις εξής παραμέτρους:

- **catsLeft**: Αριθμός γατών στην αριστερή πλευρά του ποταμού.
- **miceLeft**: Αριθμός ποντικιών στην αριστερή πλευρά του ποταμού.
- **catsRight**: Αριθμός γατών στη δεξιά πλευρά του ποταμού.
- **miceRight**: Αριθμός ποντικιών στη δεξιά πλευρά του ποταμού.
- **boatLeft**: Δείκτης που καθορίζει αν το σκάφος βρίσκεται στην αριστερή πλευρά (1 αν ναι, 0 αν όχι).
- **boatRight**: Δείκτης που καθορίζει αν το σκάφος βρίσκεται στη δεξιά πλευρά (1 αν ναι, 0 αν όχι).
- **boatCapacity**: Η μέγιστη χωρητικότητα του σκάφους (πλήθος ζώων που μπορούν να επιβιβαστούν).

Μια κατάσταση αναπαρίσταται ως εξής:



Όπου οι δύο πλευρές του ποταμού χωρίζονται με το “||” , η βάρκα στην πλευρά που υπάρχει αναπαρίσταται με το “[]” και τα ζεύγη τιμών δείχνουν το πλήθος κάθε είδους σε κάθε όχθη.

Ορίζουμε ως:

Αρχική κατάσταση : Όλοι βρίσκονται αριστερά και η βάρκα αριστερά και κανένας δεξιά.

Τελική κατάσταση : Όλοι βρίσκονται δεξιά και η βάρκα δεξιά και κανένας αριστερά.

Τελεστές Μετάβασης

Οι **τελεστές μετάβασης** καθορίζουν τις έγκυρες αλλαγές από μία κατάσταση σε άλλη.

Στην συγκεκριμένη μοντελοποίηση ορίσαμε έναν τελεστή μετάβασης , τον `loadBoat(cats, mice)` ο οποίος μας οδηγεί από μια έγκυρη κατάσταση σε μια άλλη έγκυρη κατάσταση, ικανοποιώντας τους παρακάτω περιορισμούς:

1. Φορτώνει τις επιθυμητές γάτες και ποντίκια στη.
2. Μετακινεί τη βάρκα στην άλλη πλευρά του ποταμού.
3. Ενημερώνει τον αριθμό των γατών και των ποντικιών στις δύο πλευρές του ποταμού.
4. Ενημερώνει τη θέση του σκάφους (leftSide , rightSide).

Περιορισμοί:

1. Υπάρχει όριο στον αριθμό των γατών και ποντικιών που μπορούν να μπουν στη βάρκα (`boatCapacity`)
2. Ο αριθμός των ζώων που περνάει απέναντι , δεν μπορεί να είναι μεγαλύτερος από τον υπάρχον αριθμό στην εκάστοτε όχθη (πχ για την αριστερή πλευρά `cats > this.getCatsLeft() || mice > this.getMiceLeft()`
3. Η βάρκα μπορεί να μεταφέρει γάτες και ποντίκια από την αριστερή πλευρά (leftSide) στην δεξιά (rightSide) και αντίστροφα, με την προϋπόθεση ότι η βάρκα δεν ξεπερνά τη χωρητικότητά της M(`cats + mice <= this.getBoatCapacity()`).
4. Σε κάθε όχθη αλλά και στη βάρκα, ο αριθμός των γατών δε μπορεί να ξεπερνάει τον αριθμό των ποντικών (`cats<= mice && mice>0`) (restrictions 4&5)
5. Η βάρκα δεν μπορεί να μεταφέρεται άδεια.

3. Εφαρμογή A για την Αναζήτηση της Βέλτιστης Λύσης*

Ο αλγόριθμος A* χρησιμοποιεί τη συνάρτηση **f(n) = g(n) + h(n)** για να καθοδηγήσει την αναζήτηση:

- **g(n)**: Το κόστος για να φτάσουμε στην κατάσταση n από την αρχική κατάσταση.
- **h(n)**: Η εκτίμηση του κόστους για να φτάσουμε στον στόχο από την κατάσταση n
  (αριθμός των μετακινήσεων).

Ο χώρος καταστάσεων αυξάνεται γρήγορα λόγω των πολλών πιθανών καταστάσεων και μεταβάσεων, αλλά η χρήση του αλγορίθμου A* εξασφαλίζει ότι η αναζήτηση περιορίζεται στα πιο υποσχόμενα μονοπάτια με βάση την εκτίμηση κόστους.

4.Επεξήγηση κώδικα

2.1 Κλάση Main

Το πλήθος των γατών, ποντικών (N), η χωρητικότητα της βάρκας (M) και το μέγιστο πλήθος διασχίσεων (K) , μπορούν να δοθούν ως arguments στην εκτέλεση του προγράμματος. Αν δεν εισαχθούν arguments , οι τιμές είναι οι προκαθορισμένες :

```
int N = 3; /* number of cats/mice */
int M = 2; /* max boat capacity */
int K = 100; /* max number of river-crossings */
```

Η κλάση Main δημιουργεί την αρχική κατάσταση:

```
State initialState = new State(N,N,1,0,0,0,M)
```

(γάτες , ποντίκια και βάρκα βρίσκονται στην αριστερή πλευρά).

Δημιουργούμε νέο χώρο αναζήτησης και η τελική κατάσταση είναι αυτό που επιστρέφει ο AStarClosedSet με παράμετρο την αρχική κατάσταση. Αν η τελική κατάσταση είναι null , ο αλγόριθμος απέτυχε να βρει λύση. Διαφορετικά, τυπώνεται το μονοπάτι μέχρι την τελική κατάσταση. Ξεκινώντας από την τελική κατάσταση και ακολουθώντας τις συνδέσεις μεταξύ των καταστάσεων , φτάνουμε μέχρι τη ρίζα του δέντρου (root.getFather == null). Αντιστρέφουμε το μονοπάτι και το τυπώνουμε.

2.2 Κλάση State

Η κλάση **State** αναπαριστά μία κατάσταση του προβλήματος. Κάθε κατάσταση περιγράφει τη διάταξη των γατών, των ποντικιών και της βάρκας στις δύο πλευρές του

ποταμού. Περιλαμβάνει μεθόδους για τον υπολογισμό του κόστους και της εκτίμησης του κόστους, την αναγνώριση έγκυρων μεταβάσεων, και την αποθήκευση των σχέσεων father-child για την αναπαράσταση της πορείας.

- **f, g, h**: Χρησιμοποιούνται για την εκτίμηση του κόστους της κατάστασης.
- **father**: Η κατάσταση από την οποία προήλθε η τρέχουσα κατάσταση.

Συναρτήσεις στην State

1. **loadBoat(int cats, int mice)**: Ελέγχει αν η μετάβαση είναι έγκυρη σύμφωνα με τους περιορισμούς του προβλήματος και εκτελεί τη μεταφορά από την αριστερή στην δεξιά πλευρά του ποταμού (ή αντίστροφα), ενημερώνοντας τις αντίστοιχες τιμές.
2. **getChildren()**: Δημιουργεί όλες τις δυνατές επόμενες καταστάσεις που προκύπτουν από τη φόρτωση διαφορετικών συνδυασμών γατών και ποντικιών στη βάρκα και καλεί τον τελεστή μετάβασης για να ελέγξει αν ο κάθε παραγόμενος συνδυασμός είναι έγκυρος. Εάν είναι έγκυρος, η παραγόμενη κατάσταση , παίρνει ως father την κατάσταση που κάλεσε την **getChildren()**,ενημερώνονται τα κόστη της κατάστασης και προστίθεται στη λίστα από παιδιά. Τελικά η συνάρτηση θα επιστρέψει μια λίστα από έγκυρα παιδιά.
3. **heuristic()**: Υπολογίζει πόσες φορές χρειάζεται να γεμίσει η βάρκα για να μεταφερθούν τα ζώα δεξιά. Υπάρχει διαφοροποίηση στο κόστος, αν η βάρκα βρίσκεται αριστερά ή δεξιά. Αν η βάρκα βρίσκεται *αριστερά*, διαιρούμε τον συνολικό αριθμό των ζώων που βρίσκονται αριστερά με τη χωρητικότητα της βάρκας και παίρνουμε το “ταβάνι” της διαίρεσης. Αυτό μας δίνει πόσες διασχίσεις απομένουν μέχρι την τελική κατάσταση. Εάν η βάρκα βρίσκεται *δεξιά,* χρειάζεται μια ακόμη διάσχιση από αυτές που υπολογίστηκαν. Προφανώς στην περίπτωση που η κατάσταση είναι τελική , η τιμή της ευρετικής θα είναι μηδενική.
4. **isFinal()**: Ελέγχει αν η τρέχουσα κατάσταση είναι η τελική (όταν όλες οι γάτες και ποντίκια βρίσκονται στη δεξιά πλευρά του ποταμού και η βάρκα είναι εκεί).
5. **print()**: Εκτυπώνει την τρέχουσα κατάσταση του προβλήματος, σύμφωνα με την αναπαράσταση που ορίστηκε παραπάνω.
6. **evaluate()** : Υπολογίζει το συνολικό κόστος μιας κατάστασης , αθροίζοντας το κόστος της ευρετικής (πεδίο h) με το κόστος της g (πεδίο g). Η g μας επιστρέφει την απόσταση του τρέχοντος κόμβου από την ρίζα. Η απόσταση αυτή αυξάνεται κάθε φορά που επεκτείνεται ένας κόμβος του δέντρου αναζήτησης.
7. **compareTo(State s)**: Συγκρίνει τις καταστάσεις με βάση την τιμή της evaluate().

2.3 Κλάση SpaceSearcher

Μέλη της Κλάσης

- **frontier** (Μέτωπο αναζήτησης): Αυτή η λίστα κρατά τις καταστάσεις που δεν έχουν εξεταστεί ακόμα.
- **closedSet** (Κλειστό σύνολο): Αυτή η λίστα κρατά τις καταστάσεις που έχουν ήδη εξεταστεί, έτσι ώστε να μην επαναλαμβάνεται η ίδια κατάσταση στην αναζήτηση και να αποφεύγονται οι κυκλικές αναζητήσεις.

Μέθοδος AStarClosedSet(αναζήτηση βέλτιστης λύσης)

AStarClosedSet(State initialState , int steps): Εξετάζει τις καταστάσεις από τη λίστα του μετώπου (frontier). Παίρνει ως παραμέτρους την τρέχουσα κατάσταση και το μέγιστο πλήθος εναπομείναντων διασχίσεων.

Η μέθοδος αρχικά ελέγχει αν η τρέχουσα κατάσταση είναι ήδη η τελική. Αν είναι, επιστρέφει άμεσα την κατάσταση. Αν η κατάσταση δεν είναι τελική την προσθέτουμε στο τέλος του μετώπου αναζήτησης. Εξετάζουμε το μέτωπο ξεκινώντας από το πρώτο στοιχείο του και (εφόσον αυτό δεν είναι τελική κατάσταση και δεν περιέχεται στο κλειστό σύνολο) το προσθέτουμε στο κλειστό σύνολο και τα παιδιά του στο μέτωπο.

Ακολουθούν **output** για συγκεκριμένες εισόδους M,N...

Output για N=3 , M=2 :

```
Initial State : State{f=0, h=3, g=0, father=null, totalTime=0}
(3 cats,3 mice) [] ||  (0 cats,0 mice)
-------------------------------------------
(2 cats,2 mice)  || [] (1 cats,1 mice)
-------------------------------------------
(2 cats,3 mice) [] ||  (1 cats,0 mice)
-------------------------------------------
(0 cats,3 mice)  || [] (3 cats,0 mice)
-------------------------------------------
(1 cats,3 mice) [] ||  (2 cats,0 mice)
-------------------------------------------
(1 cats,1 mice)  || [] (2 cats,2 mice)
-------------------------------------------
(2 cats,2 mice) [] ||  (1 cats,1 mice)
-------------------------------------------
(2 cats,0 mice)  || [] (1 cats,3 mice)
-------------------------------------------
(3 cats,0 mice) [] ||  (0 cats,3 mice)
-------------------------------------------
(1 cats,0 mice)  || [] (2 cats,3 mice)
-------------------------------------------
(1 cats,1 mice) [] ||  (2 cats,2 mice)
-------------------------------------------
(0 cats,0 mice)  || [] (3 cats,3 mice)
-------------------------------------------

Search time:0.0 sec.
```

Output για N=4 , M=3 :

```
Initial State : State{f=0, h=3, g=0, father=null, totalTime=0}
(4 cats,4 mice) [] ||  (0 cats,0 mice)
-------------------------------------------
(3 cats,3 mice)  || [] (1 cats,1 mice)
-------------------------------------------
(3 cats,4 mice) [] ||  (1 cats,0 mice)
-------------------------------------------
(2 cats,2 mice)  || [] (2 cats,2 mice)
-------------------------------------------
(3 cats,3 mice) [] ||  (1 cats,1 mice)
-------------------------------------------
(3 cats,0 mice)  || [] (1 cats,4 mice)
-------------------------------------------
(4 cats,0 mice) [] ||  (0 cats,4 mice)
-------------------------------------------
(2 cats,0 mice)  || [] (2 cats,4 mice)
-------------------------------------------
(3 cats,0 mice) [] ||  (1 cats,4 mice)
-------------------------------------------
(0 cats,0 mice)  || [] (4 cats,4 mice)
-------------------------------------------

Search time:0.0 sec.
```

Output για N=10 , M=5 :

```
Initial State : State{f=0, h=4, g=0, father=null, totalTime=0}
(10 cats,10 mice) [] ||  (0 cats,0 mice)
-------------------------------------------
(9 cats,9 mice)  || [] (1 cats,1 mice)
-------------------------------------------
(9 cats,10 mice) [] ||  (1 cats,0 mice)
-------------------------------------------
(4 cats,10 mice)  || [] (6 cats,0 mice)
-------------------------------------------
(5 cats,10 mice) [] ||  (5 cats,0 mice)
-------------------------------------------
(5 cats,5 mice)  || [] (5 cats,5 mice)
-------------------------------------------
(6 cats,6 mice) [] ||  (4 cats,4 mice)
-------------------------------------------
(4 cats,4 mice)  || [] (6 cats,6 mice)
-------------------------------------------
(5 cats,5 mice) [] ||  (5 cats,5 mice)
-------------------------------------------
(5 cats,0 mice)  || [] (5 cats,10 mice)
-------------------------------------------
(6 cats,0 mice) [] ||  (4 cats,10 mice)
-------------------------------------------
(4 cats,0 mice)  || [] (6 cats,10 mice)
-------------------------------------------
(5 cats,0 mice) [] ||  (5 cats,10 mice)
-------------------------------------------
(0 cats,0 mice)  || [] (10 cats,10 mice)
-------------------------------------------

Search time:0.002 sec.
```