Μέρος Α΄

Το πρόβλημα των Κανιβάλων και Ιεραποστόλων

- *State.java:*

Το αρχείο υλοποιεί την κλάση State, η οποία περιγράφει το καλούπι κάθε κατάστασης του προς επίλυση προβλήματος. Παρέχει όλες τις πληροφορίες για κάθε κατάσταση, τη μεθοδολογία για τη δημιουργία διαδόχων, τη συνάρτηση κόστους και ευρετικής, καθώς και μεθόδους για την υποστήριξη της υλοποίησης του αλγόριθμου αναζήτησης A*.

Μεταβλητές κλάσης και Κατασκευαστές:

Το αρχείο ξεκινά με τον ορισμό των βασικών μεταβλητών κάθε αντικειμένου της κλάσης State. Πιο αναλυτικά, σε κάθε κατάσταση, περιέχονται πληροφορίες σχετικά με το πλήθος των κανιβάλων και των ιεραποστόλων στην αριστερή και στην δεξιά όχθη, την θέση της βάρκας, το κόστος (πλήθος κινήσεων/μήκος) του μονοπατιού από την αρχική έως την τρέχουσα κατάσταση, την ευρετική τιμή που εκτιμά το υπόλοιπο κόστος για την επίτευξη της τελικής κατάστασης, το συνολικό κόστος, το πλήθος των διασχίσεων της βάρκας καθώς και μια αναφορά στην προηγούμενη κατάσταση. Ο βασικός κατασκευαστής της κλάσης αρχικοποιεί τις παραπάνω μεταβλητές ενώ υπάρχει και ένας κατασκευαστής αντιγραφής που καλεί τον βασικό κατασκευαστή της κλάσης αντιγράφοντας όλες τις τιμές του αντικειμένου που του έχει δοθεί ως όρισμα.

Μέθοδος isFinalState():

Η μέθοδος αυτή ελέγχει αν μια κατάσταση είναι τελική. Σκοπός του προβλήματος είναι να μεταφερθούν όλοι οι κανίβαλοι και οι ιεραπόστολοι από την αριστερή όχθη στην δεξιά. Συνεπώς μια κατάσταση είναι τελική αν και μόνο αν δεν υπάρχει κανένας κανίβαλος και κανένας ιεραπόστολος στην αριστερή όχθη (εναλλακτικά και ισοδύναμα αν όλοι οι κανίβαλοι και όλοι οι ιεραπόστολοι είναι στην δεξιά όχθη). Η μέθοδος χρησιμοποιεί την πρώτη συνθήκη τερματισμού και επιστέφει True όταν αυτή ισχύει ενώ σε κάθε άλλη περίπτωση επιστρέφει False. Η πολυπλοκότητα της μεθόδου είναι σταθερή (O(1)) καθώς κάθε φορά ελέγχει δύο ισότητες και πραγματοποιεί μια σύζευξη.

*Μέθοδος isValidState(**int** ml, **int** cl, **int** mr, **int** cr):*

Η μέθοδος αυτή ελέγχει εάν μια κατάσταση είναι έγκυρη δεδομένων των κανόνων και των τιμών του πλήθους των ιεραποστόλων και των κανιβάλων στην αριστερή και στην δεξία όχθη. Μια κατάσταση είναι έγκυρη αν και μόνο αν, τα άνω πλήθη έχουν όλα θετικές τιμές και για κάθε όχθη, αν υπάρχουν ιεραπόστολοι σε αυτή, ο αριθμός τους δεν είναι μικρότερος από το πλήθος των

Κανίβαλων της συγκεκριμένης όχθης (προστασία ιεραποστόλων). Αν ισχύουν όλα τα παραπάνω η μέθοδος επιστέφει True ενώ αν προσβάλλεται έστω και μια από τις άνω συνθήκες επιστρέφει False. Η πολυπλοκότητα της μεθόδου είναι σταθερή (O(1)), γιατί κάθε φορά πραγματοποιεί το ίδιο στατικό πλήθος συγκρίσεων.

*Μέθοδος Heuristic(**int** M):*

Η μέθοδος υπολογίζει την ευρετική τιμή εκτιμώντας το ελάχιστο πλήθος διασχίσεων που απαιτούνται για να μετακινηθούν όλοι οι κανίβαλοι και οι ιεραπόστολοι στην άλλη όχθη υποθέτωντας ότι η βάρκα πάντα θα μεταφέρει το μέγιστο δυνατό φορτίο. Αρχικά, υπολογίζει το πλήθος ιεραποστόλων και κανιβάλων που απομένουν στην αριστερή όχθη και το διαιρεί με την χωρητικότητα της βάρκας, η πράξη (remaining+M−1)/M ουσιαστικά πραγματοποιεί στρογγυλοποίηση προς τα πάνω. Η ευρετική είναι αποδεκτή καθώς γνωρίζουμε ότι το ακριβές βέλτιστο κόστος λύσης σε ένα απλοποιημένο πρόβλημα (που έχει προκύψει με αφαίρεση περιορισμών) είναι αποδεκτή ευρετική του αρχικού προβλήματος (αφού στο αρχικό πρόβλημα απαιτούνται περισσότερες ή το πολύ ίσες κινήσεις). Δηλαδή η πραγματική τιμή του κόστους (τιμή g) δεν μπορεί ποτέ να είναι μικρότερη, θα είναι το πολύ ίση με την ευριστική τιμή και στις περισσότερες περιπτώσεις μεγαλύτερη από την ευριστική. Στο κανονικό πρόβλημα επιτρέπεται να υπάρξει φορτίο βάρκας μικρότερο από την χωρητικότητα της βάρκας, κάτι το οποίο η ευριστική αγνοεί συνεπώς υποεκτιμά το συνολικό κόστος διότι το πλήθος των διασχίσεων ελαχιστοποιείται όταν κάθε φορά η βάρκα γεμίζει, ενώ αυτό δεν γίνεται στην πραγματικότητα άρα το πλήθος των διασχίσεων δεν θα είναι ποτέ το ελάχιστο δυνατό. Η ευρετική είναι συνεπής, δηλαδή για κάθε διάδοχο n', $h(n) \leq c(n,n') + h(n')$ με $c(n,n')=1$ για κάθε μετάβαση που γίνεται μεταξύ καταστάσεων. Για κάθε διάδοχο n', το h(n') υπολογίζει το ελάχιστο κόστος από τον διάδοχο μέχρι τον στόχο. Το h(n) υπολογίζει το ελάχιστο κόστος από το τρέχον n μέχρι τον στόχο. Άρα, $h(n) \leq 1 + h(n')$ το οποίο ισχύει επειδή η ευρετική βασίζεται στον αριθμό των ατόμων που απομένουν για να μεταφερθούν και στην χωρητικότητα της βάρκας. Η πολυπλοκότητα της μεθόδου είναι σταθερή (O(1)) διότι εκτελεί μια ανάθεση και 4 απλές πράξεις.

*Μέθοδος generateSuccessors(**int** M, **int** K):*

Η μέθοδος παράγει όλες τις έγκυρες δυνατές καταστάσεις που μπορούν να προκύψουν ανάλογα με την χωρητικότητα της βάρκας, τον μέγιστο επιτρεπτό αριθμό διασχίσεων (παράμετροι μεθόδου) και τις κινήσεις που μπορεί να πραγματοποιήσει, σύμφωνα με τους κανόνες του προβλήματος. Αρχικά, δημιουργεί μία κενή λίστα για να αποθηκεύσει τις νέες καταστάσεις (successors) που θα προκύψουν. Το διπλό for loop, ελέγχει όλους τους πιθανούς

συνδυασμούς αριθμού Ιεραποστόλων (m) και Κανιβάλων (c) που μπορούν να μπουν στη βάρκα. Κάθε φορά ελέγχεται η εγκυρότητα του φορτίου της βάρκας, δηλαδή αν υπάρχει τουλάχιστον ένα άτομο πάνω σε αυτή και αν το άθροισμα του πλήθους των κανιβάλων και των ιεραποστόλων δεν υπερβαίνει την χωρητικότητα της βάρκας. Επίσης, σε περίπτωση που υπάρχει πάνω από ένας Ιεραπόστολος στη βάρκα, ελέγχεται αν το πλήθος των Κανιβάλων στην βάρκα υπερβαίνει το πλήθος το Ιεραποστόλων, και αν αυτό ισχύει ο συνδυασμός αυτός απορρίπτεται και συνεχίζεται ο έλεγχος με τον επόμενο συνδυασμό. Αν η βάρκα βρίσκεται στην αριστερή όχθη τότε θα μεταβεί στην δεξιά συνεπώς για να υπολογιστεί το νέο πλήθος ιεραποστόλων και κανιβάλων σε κάθε όχθη, αφαιρείτε το πλήθος των προς μεταφορά ιεραποστόλων και κανιβάλων από τα αντίστοιχα πλήθη της αριστερής όχθης και προστίθεται στα πλήθη της δεξιάς όχθης. Με τις νέες τιμές που έχουν δημιουργηθεί και κάνοντας χρήση της μεθόδου *isValidState* ελέγχεται αν οι τιμές αυτές δημιουργούν μια έγκυρη κατάσταση, ενώ ταυτόχρονα ελέγχεται εάν η συγκεκριμένη κατάσταση, που αυξάνει κατά ένα τον αριθμό των διασχίσεων, δεν υπερβαίνει το μέγιστο επιτρεπτό όριο διασχίσεων. Αν τα προηγούμενα ισχύουν, τότε η κατάσταση δημιουργείται χρησιμοποιώντας τις άνω τιμές, αυξάνοντας το κόστος του μονοπατιού και των διασχίσεων κατά ένα και υπολογίζοντας την ευριστική τιμή κάνοντας χρήση της μεθόδου *Heuristic*. Η τρέχουσα κατάσταση αποτελεί τον γονέα της νέας, συνεπώς ενημερώνεται και η αντίστοιχη μεταβλητή της νέας κατάστασης, η οποία προστίθεται στην λίστα που έχει δημιουργηθεί στην αρχή της μεθόδου. Τα αντίστοιχα βήματα εκτελούνται αν η βάρκα βρίσκεται στην δεξιά όχθη και μεταβαίνει στην αριστερή μόνο που προσαρμόζονται κατάλληλα οι αφαιρέσεις. Τέλος, επιστρέφεται η λίστα με όλες τις πιθανές έγκυρες καταστάσεις που παράχθηκαν από την τρέχουσα κατάσταση. Η πολυπλοκότητα της μεθόδου αναλύεται ως εξής. Κάθε for loop τρέχει M φορές, άρα το διπλό for loop είναι O(M^2), ενώ κάθε άλλος έλεγχος, πράξη, εκχώρηση και πολυπλοκότητα μεθόδου που καλείται είναι O(1). Συνεπώς η τελική πολυπλοκότητα της μεθόδου καθορίζεται από το διπλό for loop και είναι της τάξης O(M^2) (τετραγωνική ως προς το μήκος της εισόδου της μεθόδου).

Μέθοδος equals(Object obj):

Η μέθοδος αυτή ελέγχει αν δύο καταστάσεις είναι ίδιες, συγκρίνοντας μεταξύ τους τις τιμές όλων των βασικών μεταβλητών κάθε κατάστασης δηλαδή του πλήθους των ιεραποστόλων και των κανιβάλων σε κάθε όχθη καθώς και την όχθη που βρίσκεται η βάρκα. Οι υπόλοιπες μεταβλητές δεν χρησιμοποιούνται στην σύγκριση διότι ενδέχεται να είναι διαφορετικές εφόσον κάποια κατάσταση έχει παραχθεί από άλλο γονέα. Συνεπώς θα έδιναν εσφαλμένο αποτέλεσμα αν χρησιμοποιούνταν στην σύγκριση. Η πολυπλοκότητα της μεθόδου είναι σταθερή (O(1)) διότι κάθε φορά πραγματοποιεί το ίδιο πλήθος συγκρίσεων.

- *AStarClosedSet.java:*

Το αρχείο υλοποιεί την κλάση AStarClosedSet, η οποία περιέχει μόνο την στατική μέθοδο solve, που λαμβάνει ως παραμέτρους την αρχική κατάσταση καθώς και την χωριτικότητα της βάρκας, και υλοποιεί τον αλγόριθμο αναζήτησης A* με κλειστό σύνολο. Για την υλοποίηση του αλγορίθμου, χρησιμοποιούνται δύο βασικές δομές δεδομένων που αναπαριστούν το μέτωπο αναζήτησης και το κλειστό σύνολο. Το μέτωπο αναζήτησης αναπαρίσταται από μια ουρά προτεραιότητας στην οποία οι καταστάσεις είναι διατεταγμένες κατά αυξανόμενο f=g+h (όπου g: κόστος μέχρι την τρέχουσα κατάσταση, h: εκτίμηση κόστους μέχρι την τελική κατάσταση). Για την διάταξη των καταστάσεων χρησιμοποιείται η μέθοδος compareTo της κλάσης State. Το κλειστό σύνολο αναπαρίσταται από ένα HashSet λόγω της ευελιξίας αναζήτησης και εισαγωγής (O(1)) καθώς και της μη-ανάγκης ελέγχου ύπαρξης διπλότυπου στοιχείου λόγω του ότι το HashSet είναι ένα σύνολο. Όσο υπάρχουν καταστάσεις στο μέτωπο αναζήτησης, αφαιρούμαι και εξετάζουμε την κατάσταση με την μικρότερη τιμή f. Ελέγχουμε αν η κατάσταση που αφαιρέθηκε είναι τελική με χρήση της μεθόδου *isFinalState* της κλάσης State και αν είναι τότε επιστρέφουμε την προς εξέταση κατάσταση και στη συνέχεια τερματίζεται ο αλγόριθμος. Εάν δεν είναι τελική, προσθέτουμε την κατάσταση στο κλειστό σύνολο, το hashCode της οποίας υπολογίζεται από την ομώνυμη μέθοδο της κλάσης State, εφόσον αυτό υλοποιείται με HashSet δεν χρειάζεται να ελέγξουμε εάν υπάρχει ήδη η κατάσταση στο κλειστό σύνολο, διότι εάν υπάρχει απλά δεν προστίθεται τίποτα σε αυτό. Στη συνέχεια, δημιουργούμε τους διαδόχους της τρέχουσας κατάστασης με χρήση της μεθόδου *generateSuccessors* της κλάσης State. Μετά, για κάθε διάδοχο που έχει δημιουργηθεί, ελέγχουμε εάν υπάρχει στο κλειστό σύνολο και αν υπάρχει τότε τον παραλείπουμε και προχωράμε στον επόμενο. Εάν δεν υπάρχει, ψάχνουμε να δούμε εάν υπάρχει στο μέτωπο αναζήτησης, ελέγχοντας για ισότητα μεταξύ καταστάσεων κάνοντας χρήση της μεθόδου *equals* της κλάσης State. Σε περίπτωση που δεν υπάρχει τον προσθέτουμε. Εάν υπάρχει, στο μέτωπο, ελέγχουμε τις τιμές g (κόστος διαδρομής) των δύο καταστάσεων, αν η τιμή g του διαδόχου είναι μικρότερη από την τιμή g της κατάστασης που υπάρχει ήδη στο μέτωπο τότε αντικαθιστούμε τις δύο -ίδιες- καταστάσεις διότι πάντα θέλουμε να εξερευνούμε τις καταστάσεις με την μικρότερη τιμή g και εφόσον υπάρχει μια ίδια κατάσταση με μικρότερη τιμή g από αυτή που έχουμε, θα είναι σίγουρα προτιμότερη. Μετά το πέρας όλων των άνω ελέγχων και αφού τερματίσει ο κύριος βρόχος χωρίς να έχει επιστραφεί κάποια κατάσταση, επιστρέφεται null διότι δεν βρέθηκε καμία λύση. Εφόσον δείξαμε ότι η ευρετική που χρησιμοποιούμε είναι συνεπής, ο αλγόριθμος θα είναι βέλτιστος, δηλαδή η λύση που θα βρίσκει, αν υπάρχει, θα είναι βέλτιστη.

Πολυπλοκότητα:

Για την υλοποίηση του αλγόριθμου A* έχουν χρησιμοποιηθεί δύο βασικές δομές δεδομένων. Το HashSet χρησιμοποιείται για την προσομοίωση του κλειστού συνόλου. Το πλεονέκτημά του είναι πως εφόσον αποτελεί μια μορφή συνόλου δεν επιτρέπει τα διπλότυπα και οι πράξεις εισαγωγής και αναζήτησης έχουν μέση πολυπλοκότητα O(1). Η ουρά προτεραιότητας ελαχίστου χρησιμοποιείται για την προσομοίωση του μετώπου αναζήτησης. Οι πράξεις εισαγωγής στοιχείου καθώς και εξαγωγής του ελάχιστου έχουν πολυπλοκότητα O(logn), όπου n είναι ο αριθμός των στοιχείων στην ουρά ενώ η αναζήτηση στοιχείου και η αφαίρεσή του (απαιτεί αναδιάταξη της ουράς για την διατήρηση των ιδιοτήτων της) έχουν πολυπλοκότητα τάξης O(n), όπου n είναι ο αριθμός των στοιχείων στην ουρά . Ο βασικός βρόγχος της μεθόδου εκτελείται μέχρις ότου αδειάσει η ουρά και η πολυπλοκότητά του εξαρτάται από το πλήθος των συνολικών καταστάσεων που εξετάζονται. Κάθε έλεγχος μέσα στο While, απαιτεί O(1) χρόνο ενώ η *generateSuccessors* της κλάσης State απαιτεί O(M^2), όπου M η χωρητικότητα της βάρκας. Θεωρούμε ότι κάθε κατάσταση έχει κατά μέοσο όρο S διαδόχους και ότι συνολικά έχουμε N καταστάσεις στο μέτωπο αναζήτησης.

Σε κάθε επανάληψη του βρόγχου έχουμε τα εξής:

Α) Εξαγωγή στοιχείου από την ουρά: O(logn).

Β) Δημιουργία διαδόχων: O(M^2).

Γ) Έλεγχος διαδόχων στο κλειστό σύνολο: O(S)

Δ) Αναζήτηση στο μέτωπο: O(S·n).

Ε) Αφαίρεση και Προσθήκη στο μέτωπο : O(S·(logn+n)).

Συνολικά ανά επανάληψη:

O(logn+M^2+S·n+S·(logn+n))

Συνολικά για Όλες τις Καταστάσεις:

O(N(logn+M^2+S·n+S·(logn+n)))

- *Main.java:*

Η κλάση αυτή, περιλαμβάνει το κυρίως πρόγραμμα που υλοποιεί το πρόβλημα των Κανιβάλων και Ιεραποστόλων. Αρχικά, μετατρέπει τις 3 παραμέτρους εισόδου (Αριθμός Ιεραποστόλων/Κανιβάλων, Μέγιστη Χωριτικότητα Βάρκας, Μέγιστο πλήθος Διασχίσεων) σε μορφή ακεραίου για να μπορέσουν να χρησιμοποιηθούν στην περαιτέρω επεξεργασία. Στην συνέχεια δημιουργείται η αρχική κατάσταση που περιλαμβάνει όλους τους Κανιβάλους και τους Ιεραποστόλους στην αριστερή όχθη καθώς και όλα τα άλλα ορίσματα ίσα με 0. Πριν εκτελεστεί ο αλγόριθμος αναζήτησης, καθώς και αφού έχει τερματίσει, καταγράφεται η ώρα του συστήματος. Εφόσον ο αλγόριθμος αναζήτησης έχει ολοκληρωθεί, ελέγχεται εάν υπάρχει λύση, αν δεν υπάρχει λύση τότε εμφανίζεται κατάλληλο μήνυμα, ενώ αν υπάρχει εκτυπώνεται το μονοπάτι της λύσης με την χρήση της μεθόδου *printSolution*. Τέλος, ανεξάρτητα με τον αν υπήρξε λύση ή όχι, υπολογίζεται και εμφανίζεται ο χρόνος που χρειάστηκε ο A* για να αποκριθεί.

printSolution(State state):

Η μέθοδος αυτή καλείται μόνο εάν έχει βρεθεί λύση, δηλαδή η μέθοδος που υλοποιεί τον A* έχει επιστρέψει μια κατάσταση, και χρησιμοποιείται για να βρεί το μονοπάτι της λύσης. Ξεκινώντας από την τελική κατάσταση, προσθέτει την τρέχουσα κατάσταση σε ένα ArrayList και κάνοντας χρήση της μεθόδου *getParent* της κλάσης State ανακτά τον πατέρα της, δηλαδή τον προηγούμενο κόμβο στο μονοπάτι της λύσης. Η άνω επαναληπτική διαδικασία πραγματοποιείται μέχρι να βρεθεί κατάσταση ίση με *null*, δηλαδή μέχρι να ανακτηθεί το μονοπάτι από την τελική στην αρχική κατάσταση. Εφόσον το μονοπάτι έχει βρεθεί ανάποδα, χρησιμοποιείται η μέθοδος *reverse* για να το αντιστρέψει. Τέλος, κάθε κόμβος στο μονοπάτι εκτυπώνεται, καλώντας την υπερφορτωμένη μέθοδο *toString* της κλάσης State.

Παράμετρος Μέγιστου Αριθμού Διασχίσεων (K):

Η παράμετρος αυτή χρησιμοποιείται απευθείας στην μέθοδο *generateSuccessors* της κλάσης State. Πριν δημιουργηθεί κάποια κατάσταση, ελέγχεται εάν αυτή δεν ξεπερνά τον μέγιστο αριθμό διασχίσεων και εάν αυτό ισχύει τότε δεν δημιουργείται. Αν και μπορεί διαισθητικά να μην ταιριάζει η συμπερίληψη της συγκεκριμένης παραμέτρου στην μέθοδο της παραγωγής απογόνων μιας κατάστασης, απλοποιεί πολύ την λογική του ελέγχου και της πολυπλοκότητας της υπόλοιπης υλοποίησης. Εάν ο έλεγχος της παραμέτρου αυτής γινόταν στον αλγόριθμο αναζήτησης, ενδεχομένως θα παράγονταν καταστάσεις που υπερβαίνουν το μέγιστο όριο. Θα έπρεπε στη συνέχεια να ελέγχονται έτσι ώστε να απορρίπτονταν, αυξάνοντας έτσι την πολυπλοκότητα και το πλήθος των

βημάτων του αλγορίθμου. Ανάλογα, στο κυρίως πρόγραμμα θα έπρεπε, εφόσον είχε επιστραφεί λύση να ελεγχθεί το πλήθος διασχίσεων της τελευταίας κατάστασης, και εφόσον αυτό υπερέβαινε το μέγιστο όριο, η λύση να απορρίπτονταν. Και στις δύο περιπτώσεις, θα δημιουργούσαμε καταστάσεις που πρακτικά θα ήταν άχρηστες και θα καθυστερούσαμε πολύ την εκτέλεση της αναζήτησης, αναγκάζοντάς την σε περιττούς ελέγχους. Με αυτό τον τρόπο, δεν δημιουργούνται τέτοιες καταστάσεις, και εφόσον σε κάθε επόμενη κατάσταση ο αριθμός των διασχίσεων αυξάνεται, από κάποια κατάσταση και μετά δεν θα δημιουργούνταν απόγονοι, οπότε θα υπάρχουν λιγότερες καταστάσεις για να επεξεργαστεί ο αλγόριθμος αναζήτησης.

Πειραματικές Εκτελέσεις για διάφορες τιμές των παραμέτρων:

- *Για N = 3, M = 2, K = 100*

Ο αλγόριθμος βρίσκει την παρακάτω λύση, πραγματοποιώντας 11 διασχίσεις με χρόνο αναζήτησης 0.002 seconds

State{Missionaries on Left bank=3, Cannibals on Left bank=3, Missionaries on Right bank=0, Cannibals on Right bank=0, Boat=Left, g=0, h=0, f=0, crossings=0}

State{Missionaries on Left bank=2, Cannibals on Left bank=2, Missionaries on Right bank=1, Cannibals on Right bank=1, Boat=Right, g=1, h=3, f=4, crossings=1}

State{Missionaries on Left bank=3, Cannibals on Left bank=2, Missionaries on Right bank=0, Cannibals on Right bank=1, Boat=Left, g=2, h=2, f=4, crossings=2}

State{Missionaries on Left bank=3, Cannibals on Left bank=0, Missionaries on Right bank=0, Cannibals on Right bank=3, Boat=Right, g=3, h=3, f=6, crossings=3}

State{Missionaries on Left bank=3, Cannibals on Left bank=1, Missionaries on Right bank=0, Cannibals on Right bank=2, Boat=Left, g=4, h=2, f=6, crossings=4}

State{Missionaries on Left bank=1, Cannibals on Left bank=1, Missionaries on Right bank=2, Cannibals on Right bank=2, Boat=Right, g=5, h=2, f=7, crossings=5}

State{Missionaries on Left bank=2, Cannibals on Left bank=2, Missionaries on Right bank=1, Cannibals on Right bank=1, Boat=Left, g=6, h=1, f=7, crossings=6}

State{Missionaries on Left bank=0, Cannibals on Left bank=2, Missionaries on Right bank=3, Cannibals on Right bank=1, Boat=Right, g=7, h=2, f=9, crossings=7}

State{Missionaries on Left bank=0, Cannibals on Left bank=3, Missionaries on Right bank=3, Cannibals on Right bank=0, Boat=Left, g=8, h=1, f=9, crossings=8}

State{Missionaries on Left bank=0, Cannibals on Left bank=1, Missionaries on Right bank=3, Cannibals on Right bank=2, Boat=Right, g=9, h=2, f=11, crossings=9}

State{Missionaries on Left bank=0, Cannibals on Left bank=2, Missionaries on Right bank=3, Cannibals on Right bank=1, Boat=Left, g=10, h=1, f=11, crossings=10}

State{Missionaries on Left bank=0, Cannibals on Left bank=0, Missionaries on Right bank=3, Cannibals on Right bank=3, Boat=Right, g=11, h=1, f=12, crossings=11}

- *Για N = 50, M = 10, K = 100*

Ο αλγόριθμος βρίσκει την παρακάτω λύση, πραγματοποιώντας 25 διασχίσεις με χρόνο αναζήτησης 0.006 seconds

State{Missionaries on Left bank=50, Cannibals on Left bank=50, Missionaries on Right bank=0, Cannibals on Right bank=0, Boat=Left, g=0, h=0, f=0, crossings=0}

State{Missionaries on Left bank=50, Cannibals on Left bank=40, Missionaries on Right bank=0, Cannibals on Right bank=10, Boat=Right, g=1, h=10, f=11, crossings=1}

State{Missionaries on Left bank=50, Cannibals on Left bank=41, Missionaries on Right bank=0, Cannibals on Right bank=9, Boat=Left, g=2, h=9, f=11, crossings=2}

State{Missionaries on Left bank=41, Cannibals on Left bank=41, Missionaries on Right bank=9, Cannibals on Right bank=9, Boat=Right, g=3, h=10, f=13, crossings=3}

State{Missionaries on Left bank=42, Cannibals on Left bank=42, Missionaries on Right bank=8, Cannibals on Right bank=8, Boat=Left, g=4, h=9, f=13, crossings=4}

State{Missionaries on Left bank=37, Cannibals on Left bank=37, Missionaries on Right bank=13, Cannibals on Right bank=13, Boat=Right, g=5, h=9, f=14, crossings=5}

State{Missionaries on Left bank=38, Cannibals on Left bank=38, Missionaries on Right bank=12, Cannibals on Right bank=12, Boat=Left, g=6, h=8, f=14, crossings=6}

State{Missionaries on Left bank=33, Cannibals on Left bank=33, Missionaries on Right bank=17, Cannibals on Right bank=17, Boat=Right, g=7, h=8, f=15, crossings=7}

State{Missionaries on Left bank=34, Cannibals on Left bank=34, Missionaries on Right bank=16, Cannibals on Right bank=16, Boat=Left, g=8, h=7, f=15, crossings=8}

State{Missionaries on Left bank=29, Cannibals on Left bank=29, Missionaries on Right bank=21, Cannibals on Right bank=21, Boat=Right, g=9, h=7, f=16, crossings=9}

State{Missionaries on Left bank=30, Cannibals on Left bank=30, Missionaries on Right bank=20, Cannibals on Right bank=20, Boat=Left, g=10, h=6, f=16, crossings=10}

State{Missionaries on Left bank=25, Cannibals on Left bank=25, Missionaries on Right bank=25, Cannibals on Right bank=25, Boat=Right, g=11, h=6, f=17, crossings=11}

State{Missionaries on Left bank=26, Cannibals on Left bank=26, Missionaries on Right bank=24, Cannibals on Right bank=24, Boat=Left, g=12, h=5, f=17, crossings=12}

State{Missionaries on Left bank=21, Cannibals on Left bank=21, Missionaries on Right bank=29, Cannibals on Right bank=29, Boat=Right, g=13, h=6, f=19, crossings=13}

State{Missionaries on Left bank=22, Cannibals on Left bank=22, Missionaries on Right bank=28, Cannibals on Right bank=28, Boat=Left, g=14, h=5, f=19, crossings=14}

State{Missionaries on Left bank=17, Cannibals on Left bank=17, Missionaries on Right bank=33, Cannibals on Right bank=33, Boat=Right, g=15, h=5, f=20, crossings=15}

State{Missionaries on Left bank=18, Cannibals on Left bank=18, Missionaries on Right bank=32, Cannibals on Right bank=32, Boat=Left, g=16, h=4, f=20, crossings=16}

State{Missionaries on Left bank=13, Cannibals on Left bank=13, Missionaries on Right bank=37, Cannibals on Right bank=37, Boat=Right, g=17, h=4, f=21, crossings=17}

State{Missionaries on Left bank=14, Cannibals on Left bank=14, Missionaries on Right bank=36, Cannibals on Right bank=36, Boat=Left, g=18, h=3, f=21, crossings=18}

State{Missionaries on Left bank=9, Cannibals on Left bank=9, Missionaries on Right bank=41, Cannibals on Right bank=41, Boat=Right, g=19, h=3, f=22, crossings=19}

State{Missionaries on Left bank=10, Cannibals on Left bank=10, Missionaries on Right bank=40, Cannibals on Right bank=40, Boat=Left, g=20, h=2, f=22, crossings=20}

State{Missionaries on Left bank=5, Cannibals on Left bank=5, Missionaries on Right bank=45, Cannibals on Right bank=45, Boat=Right, g=21, h=2, f=23, crossings=21}

State{Missionaries on Left bank=6, Cannibals on Left bank=6, Missionaries on Right bank=44, Cannibals on Right bank=44, Boat=Left, g=22, h=1, f=23, crossings=22}

State{Missionaries on Left bank=1, Cannibals on Left bank=1, Missionaries on Right bank=49, Cannibals on Right bank=49, Boat=Right, g=23, h=2, f=25, crossings=23}

State{Missionaries on Left bank=3, Cannibals on Left bank=3, Missionaries on Right bank=47, Cannibals on Right bank=47, Boat=Left, g=24, h=1, f=25, crossings=24}

State{Missionaries on Left bank=0, Cannibals on Left bank=0, Missionaries on Right bank=50, Cannibals on Right bank=50, Boat=Right, g=25, h=1, f=26, crossings=25}

- *Για N = 50, M = 10, K = 20*

Ο αλγόριθμος δεν βρίσκει λύση, ουσιαστικά έχουμε την ίδια τιμοδοσία με πριν, που χρειάστηκε 25 συγκρίσεις αλλά πλέον ο μέγιστος αριθμός διασχίσεων είναι 20. Άρα εφόσον ο αλγόριθμος αναζήτησης είναι βέλτιστος, η προηγούμενη λύση ήταν βέλτιστη. Συνεπώς δεν θα μπορούσε να υπάρχει λύση με μέγιστο πλήθος διασχίσεων μικρότερο από το βέλτιστο. Άρα ο αλγόριθμος απαντά αναμενόμενα ορθά και αποκρίνεται σε 0.005 seconds