ΙΩΑΝΝΗΣ ΧΡΙΣΤΟΣ ΜΠΑΛΤΑΣ – 3180113

ΙΩΑΝΝΗΣ ΛΟΥΛΑΚΗΣ - 3180262

ΕΡΓΑΣΙΑ 1η

Το πρόβλημα των κανιβάλων και ιεραποστόλων

Για τη λύση του προβλήματος έχουμε χρησιμοποιήσει τον αλγόριθμο A* με κλειστό σύνολο. Για την ευρετική αφαιρούμε τον περιορισμό που δεν επιτρέπει ο αριθμός των κανιβάλων σε κάποια όχθη να υπερβεί τον αριθμό των ιεραποστόλων στην ίδια όχθη. Στην περίπτωση αυτή αν ο αριθμός των ατόμων είναι v και βρισκόμαστε στην αριστερή όχθη, ομοίως και η βάρκα, τότε ο ελάχιστος αριθμός κινήσεων μέχρι το στόχο για v=1 είναι μία διάσχιση(1). Για v=2 έχουμε 2*(v-1) -1 διασχίσεις(2). Για v>2 βρήκαμε μία ευρετική που αφενός είναι αποδεκτή, αφού υποεκτιμά το πρόβλημα, διότι έχουμε αφαιρέσει τον περιορισμό που δεν επιτρέπει ο αριθμός των κανιβάλων σε κάποια όχθη να υπερβεί τον αριθμό των ιεραποστόλων στην ίδια όχθη και συνεπώς το κόστος της ευρετικής είναι σίγουρα μικρότερο του πραγματικού και αφετέρου είναι και συνεπείς, αφού από τη θεωρία γνωρίζουμε ότι όταν το κόστος κάθε μετάβασης είναι θετικό, οι ευρετικές που προκύπτουν με αφαίρεση περιορισμών είναι συνεπείς.

Η ευρετική που βρήκαμε είναι η:

Math.round(2 * (missionariesLeft + cannibalsLeft) /(float)(boatCapacity - 1) -1) και ισχύει όταν όσοι βρίσκονται στην αριστερή μεριά της όχθης είναι περισσότεροι από την χωρητικότητα της βάρκας, αλλιώς γίνεται 1 διάσχιση.

Στο αρχείο TestHeuristic.java τρέχουμε την ευρετική για M από 2 έως και 10 για όλα τα N από 2 έως και 30 και βγάζει τις διασχίσεις που γίνονται. Παρατηρούμε πως τις βρίσκει σωστά ή άντε το πολύ να απέχει +- 1.

Όπου missionariesLeft και cannibalsLeft είναι ο αρχικός αριθμός των κανιβάλων και ιεραποστόλων στην αριστερή μεριά της όχθης και boatCapacity είναι η χωρητικότητα της βάρκας.

Επίσης για g(x) έχουμε βάλει το βάθος του δέντρου.

Έχουμε υλοποιήσει τρία java αρχεία. Αρχικά, έχουμε το State στο οποίο έχουμε 2 constructor για την δημιουργία ενός αντικειμένου. Ο πρώτος καλείται στην main (Main.java) για να δημιουργηθεί την πρώτη φορά η ρίζα του δέντρου με τις αρχικές τιμές που μας

δίνονται κατά την εκτέλεση του προγράμματος. Ο 2ος constructor χρησιμοποιείται όταν δημιουργείται ένα νέο παιδί για να περάσει ουσιαστικά στην σε νέα κατάσταση με μειωμένο αριθμό ατόμων ανάλογα με όσους πήρε για να μεταφέρει στην απέναντι όχθη πάντα με γνώμονα τους περιορισμούς. Οι παράμετροι που παίρνει είναι οι missionariesLeft, cannibalsLeft, missionariesRight, cannibalsRight, boatCapacity, g, sideOfBoat. Το g όταν καλείται ο constructor αυξάνεται κατά 1, ώστε να βρίσκεται και το βάθος του δέντρου και να προστίθεται στο τελικό score. Στην συνέχεια, έχουμε μία μέθοδο print που εκτυπώνει την κατάσταση, τον κόμβο του δέντρου δηλαδή που επίλεξε ο A* ως καλύτερο, την getChildren η οποία ανάλογα με την μεριά που βρίσκεται η βάρκα, γι' αυτό έχουμε ένα enum BoatSide με δύο τιμές: left, right, καλεί τη μέθοδο goLeft ή goRight. Αυτές οι μέθοδοι ψάχνουν για κάθε συνδυασμό ιεραποστόλων και κανιβάλων, χρησιμοποιώντας μία λούπα για κάθε έναν της ανάλογης μεριάς και αν πληρούνται οι περιορισμοί τότε για τον συνδυασμό που πληρούνται φτιάχνει το παιδί και αν τελικά το παιδί είναι έγκυρο, καλεί την isValid για τη λειτουργία αυτή, καλεί την ευρετική θέτει για πατέρα το αντικείμενο αυτό και το προσθέτει το παιδί στο ArrayList children.

Η heuristic ειναι η ευρετικη μας όπου υπολογίζει το score προσθέτωντας στην h που είδαμε πιο πάνω το g και υλοποιούνται και οι equals, hashCode και compareTo για να συγκρίνονται 2 States. Η hashCode επιστρέφει το άθροισμα των τετραγώνων των κανιβάλων και ιεραποστόλων στην αριστερή πλευρά.

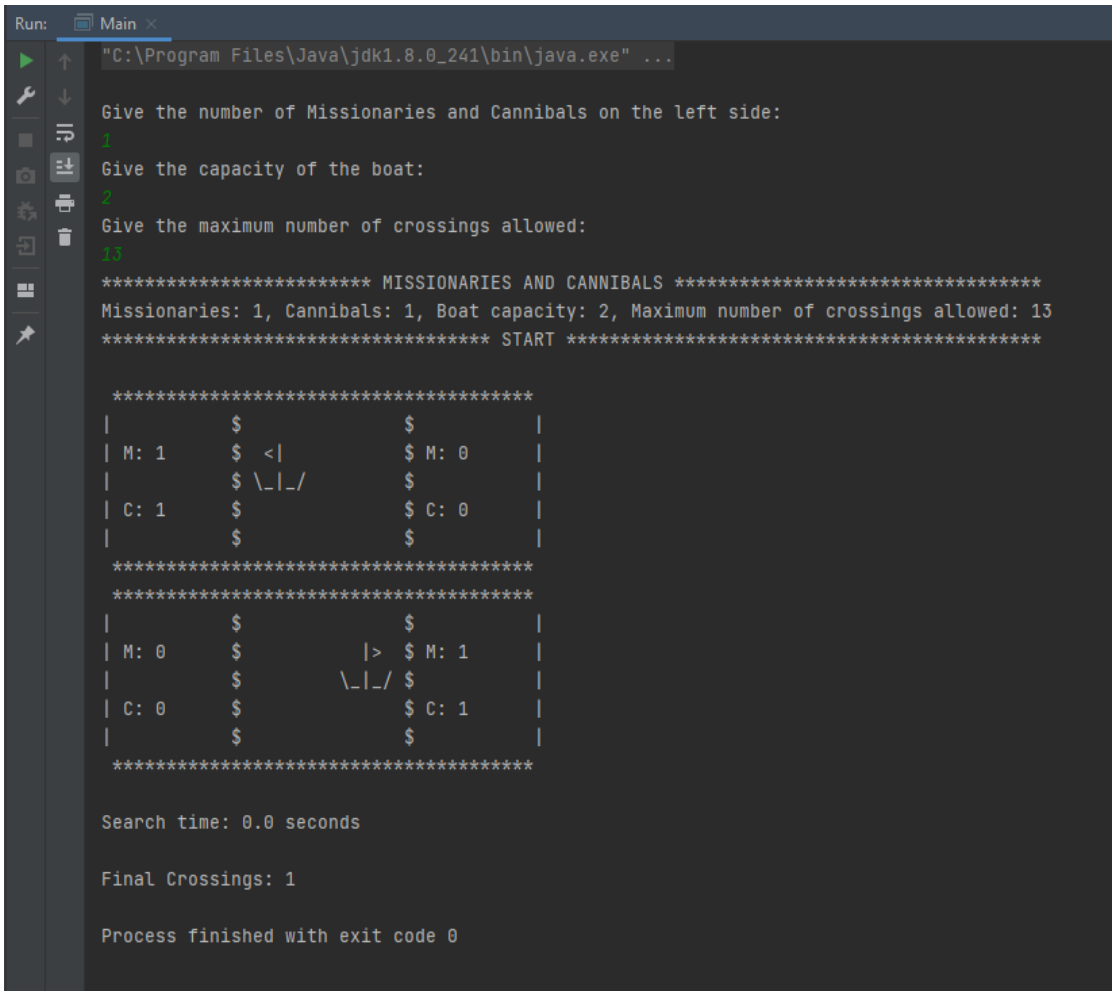
Επιπλέον, στο αρχείο AStarSearcher.java υλοποιείται ο αλγόριθμος A* με κλειστό σύνολο. Αρχικοποιείται το μέτωπο και το κλειστό σύνολο ώς ένα ArrayList και HashSet αντίστοιχα. Αρχικά ελέγχουμε αν η αρχική κατάσταση που του περνάμε στην είσοδο κλήσης του είναι η τελική κατάσταση τότε και την επιστρέφουμε. Αφαιρούμε εν συνεχεία από το μέτωπο τον πρώτο κόμβο και το θέτουμε για τωρινή κατάσταση ώστε να το ελένξουμε και αν είναι τελική κατάσταση να το επιστρέψουμε. Αν η τωρινή κατάσταση δεν βρίσκεται στο κλειστό σύνολο τότε την προσθέτουμε σε αυτό και προσθέτουμε στο μέτωπο όλα τα παιδιά του συγκεκριμένου κόμβου και ταξινομούμε το μέτωπο. Αν δεν επιστραφεί τίποτα από τα παραπάνω τότε ο αλγόριθμος επιστρέφει null.

Τέλος, στη main (Main.java) ζητούνται και διαβάζονται ο αριθμός των ιεραποστόλων και κανιβάλων στην αριστερή όχθη, η χωρητικότητα της βάρκας καθώς και ο μέγιστος επιτρεπόμενος αριθμών διασχίσεων που επιτρέπονται. Στη συνέχεια, με τα δεδομένα αυτά δημιουργούμε τη ρίζα του δέντρου και καλούμε την μέθοδο του A* ώστε να βρεί λύση, αν υπάρχει μιας που είναι και πλήρης, και να την επιστρέψει. Αν μας έχει επιστραφεί null από τον αλγόριθμο τότε σημαίναι πως δεν βρήκε λύση οπότε το εκτυπώνουμε. Αν όμως βρήκε λύση τότε βρίσκουμε το μονοπάτι από την τελική κατάσταση προς τη ρίζα το εντιστρέφουμε και το εκυπώνουμε. Μαζί με το μονοπάτι εκτυπώνεται και ο αριθμός των διασχίσεων όπως και ο χρόνος που χρειάστηκε ο αλγόριθμος να βρεί λύση. Αν ο αριθμός των μέγιστων επιτρεπόμενων διασχίσεων ξεπεράσει είναι μικρότερος από τον αριθμό διασχίσεων που βρήκε ο αλγόριθμος τότε εκτυπώνται και οι δύο και ενημερώνεται ο χρήστης ότι δυστυχώς δεν ήταν αρκετός για να βρεί λύση με λιγότερες διασχίσεις.

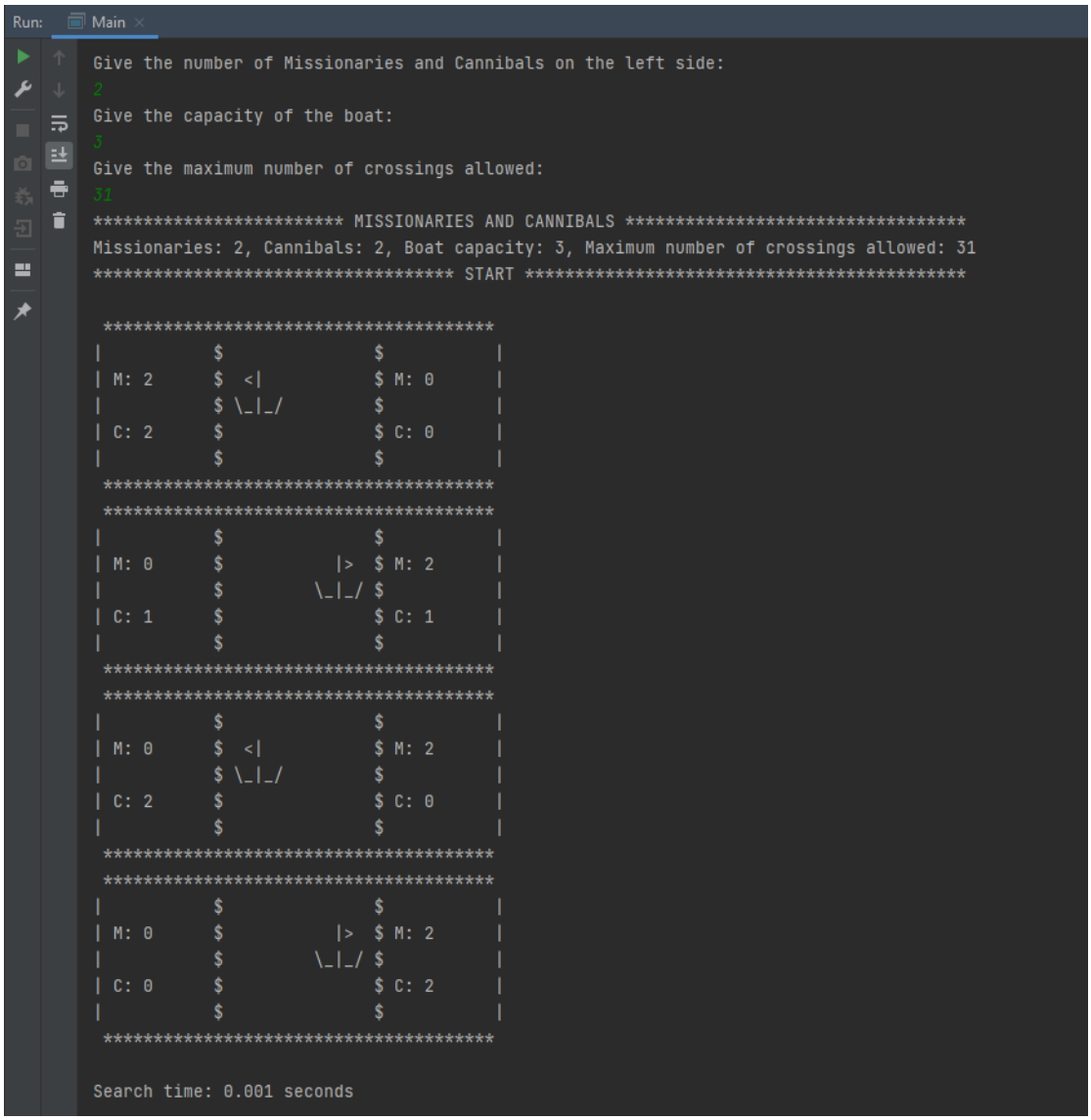
Παραδείγματα λειτουργίας (ο χρόνος φαίνεται στα screenshot):

Ο κώδικας έχει τρέξει σε υπολογιστή με AMD Ryzen 7 2700x, 16GB ram, gtx 1650, m2 ssd.

(1)N = 1, M = 2 (1 crossing όπως φαίνεται και από το σχήμα)

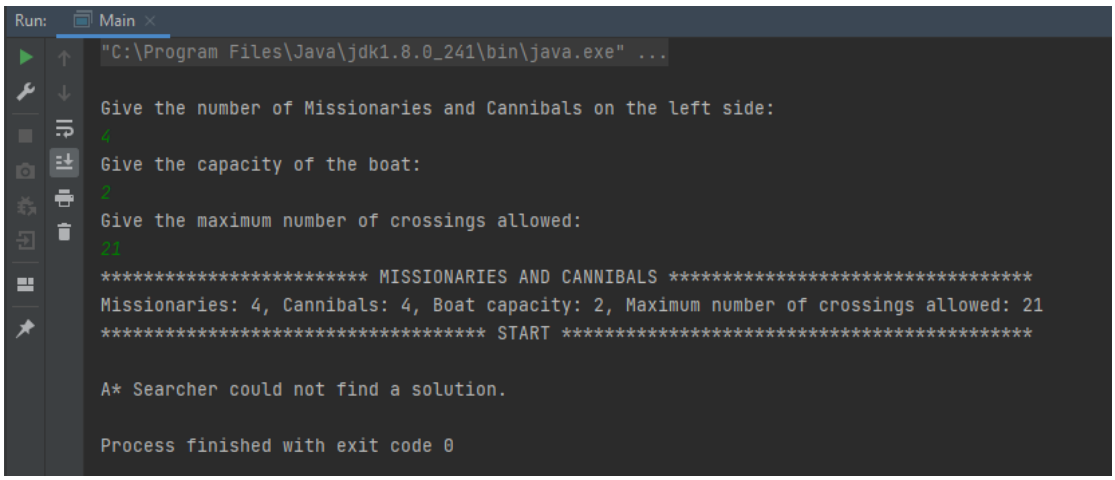

Run:
Main
"C:\Program Files\Java\jdk1.8.0_241\bin\java.exe" ...
Give the number of Missionaries and Cannibals on the left side:
1
Give the capacity of the boat:
2
Give the maximum number of crossings allowed:
13
********** MISSIONARIES AND CANNIBALS **********
Missionaries: 1, Cannibals: 1, Boat capacity: 2, Maximum number of crossings allowed: 13
********** START **********
| $ $ |
| M: 1 $ <| $ M: 0 |
| $ _|_/ $ |
| C: 1 $ $ C: 0 |
| $ $ |
| $ $ |
| M: 0 $ |> $ M: 1 |
| $ _|_/ $ |
| C: 0 $ $ C: 1 |
| $ $ |
Search time: 0.0 seconds
Final Crossings: 1
Process finished with exit code 0

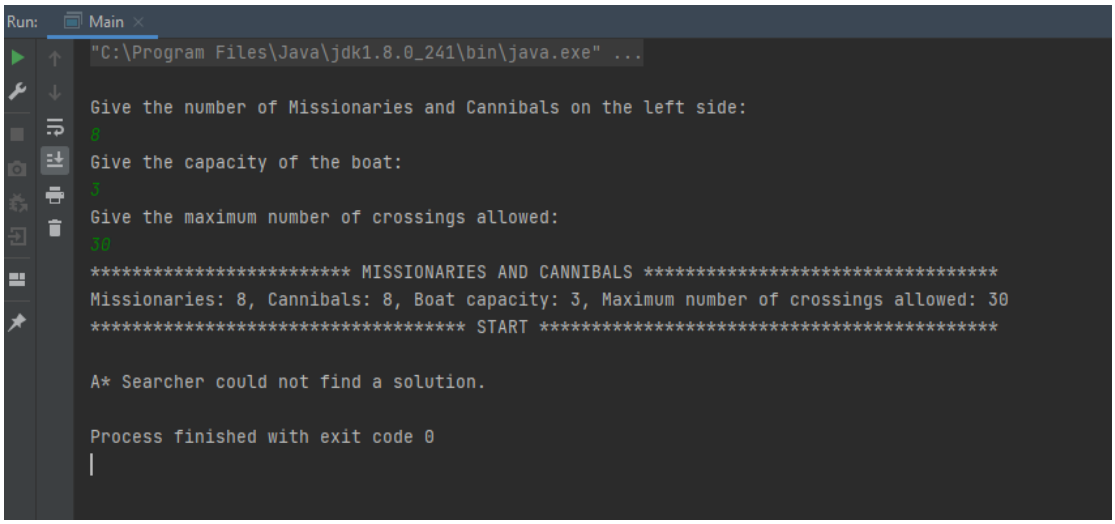
(2) N = 2, M = 3, K = 31



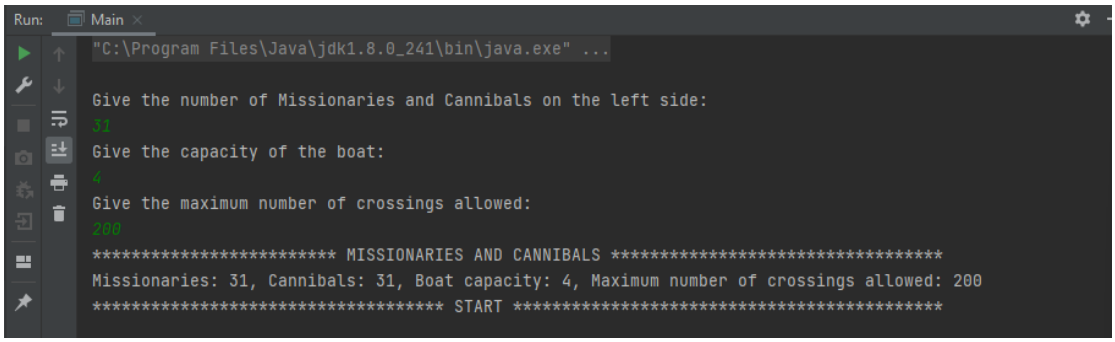

```
Final Crossings: 3

Process finished with exit code 0
```

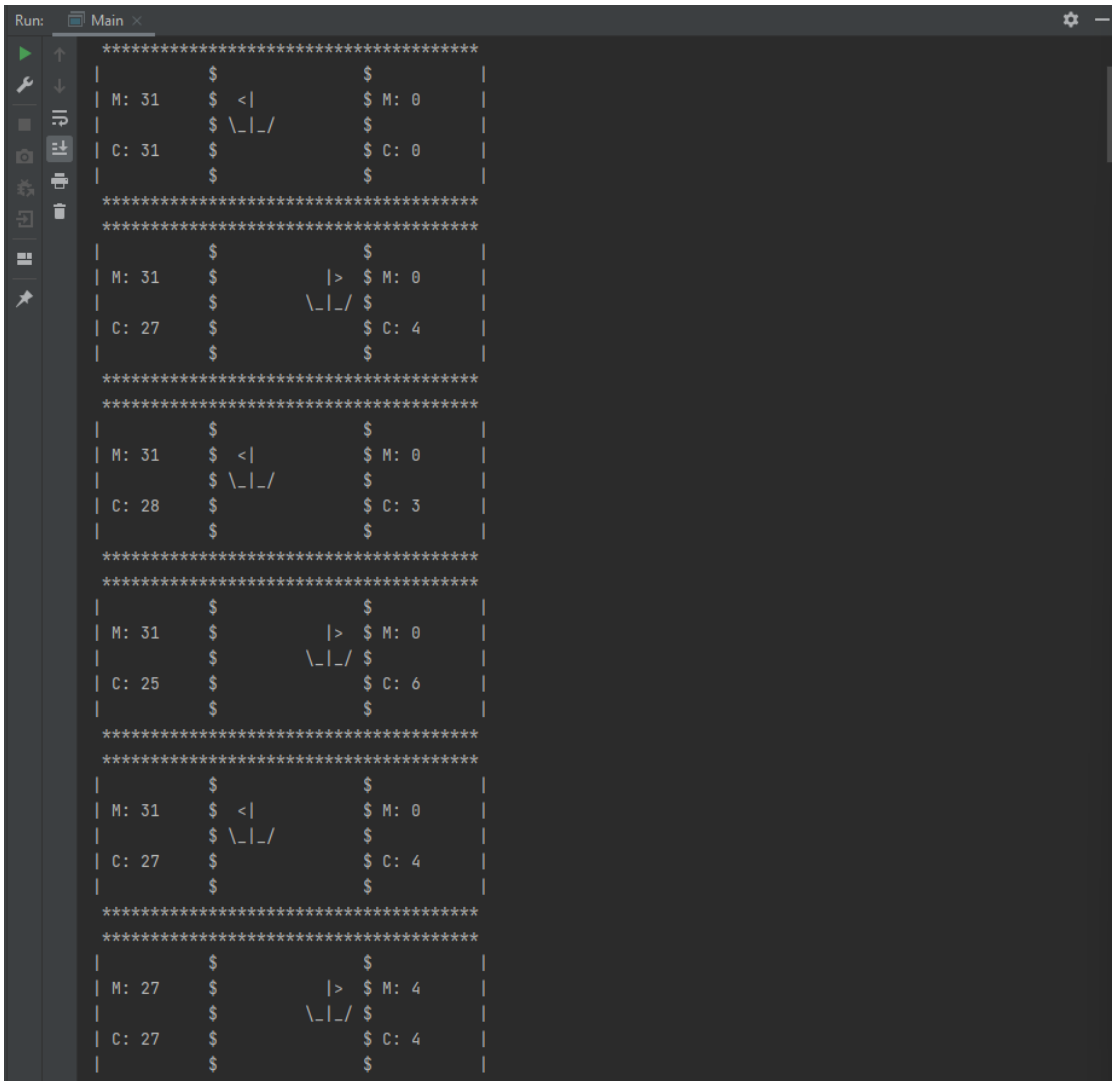
3) Για M = 2 και N>3 δεν υπάρχει λύση και όντως το βρίσκει ο αλγόριθμος, όπως και για M = 3 και N>5.



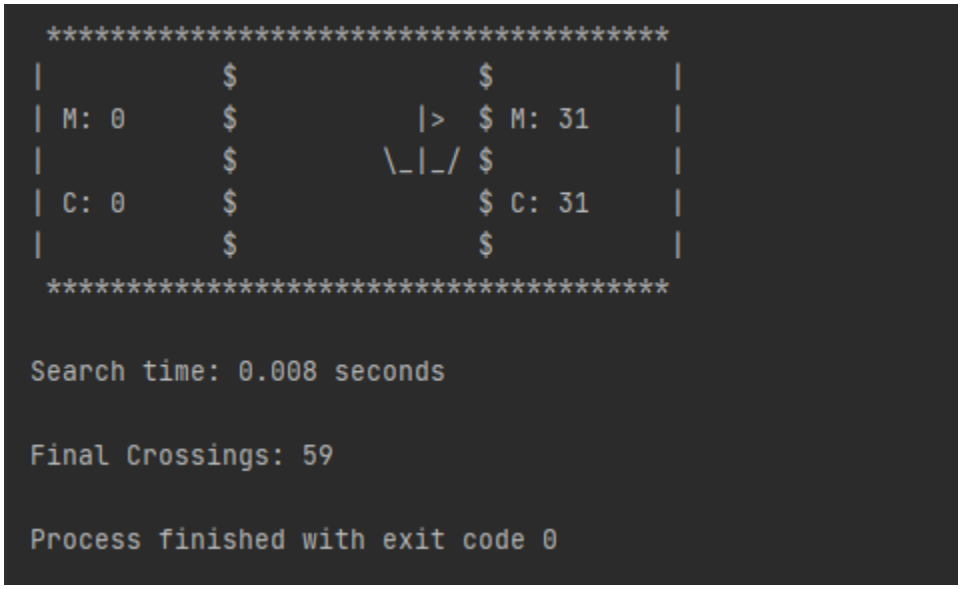


4) Για M > 3 ο αλγόριθμος βρίσκει λύση για όλα τα N.

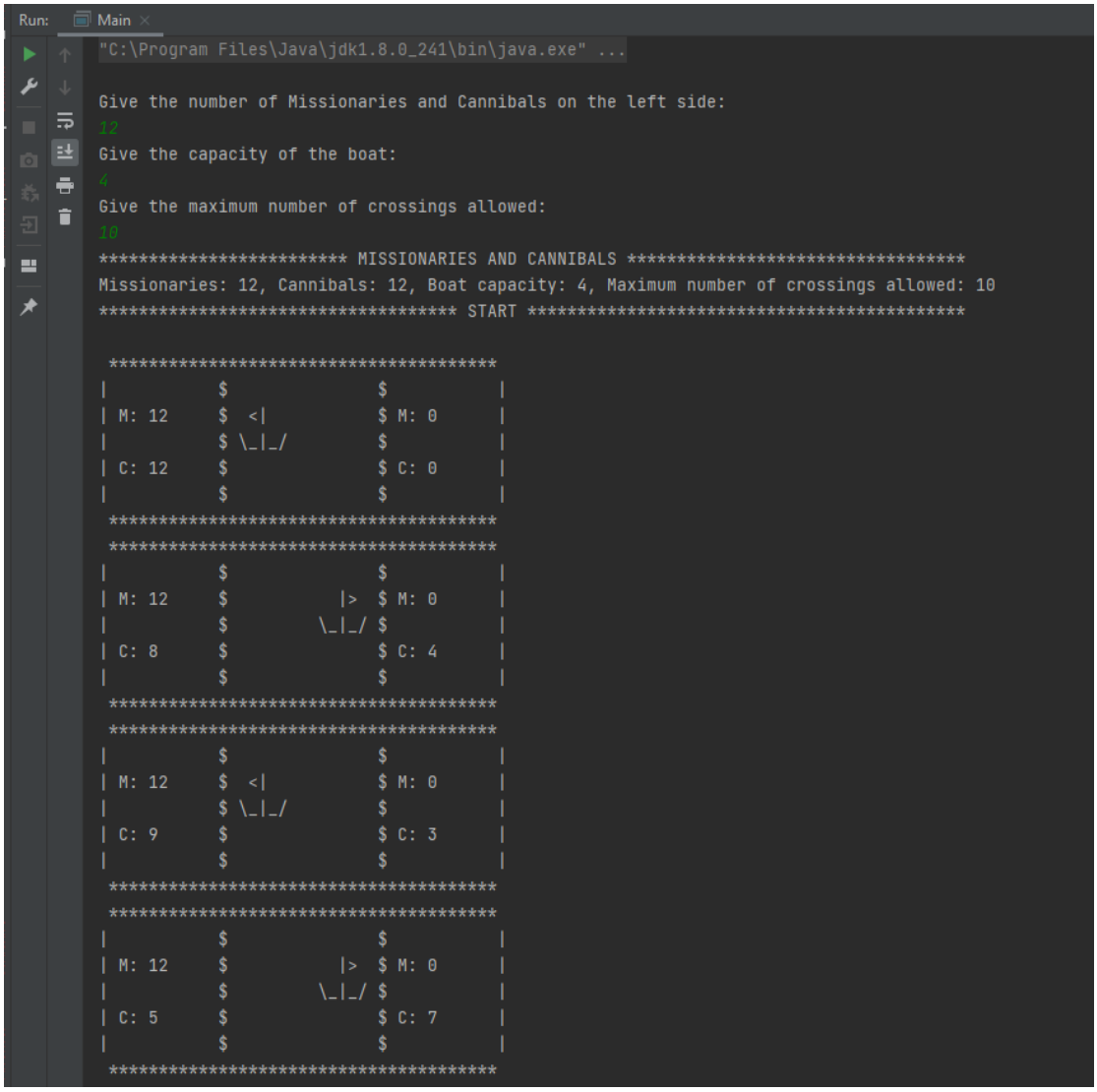


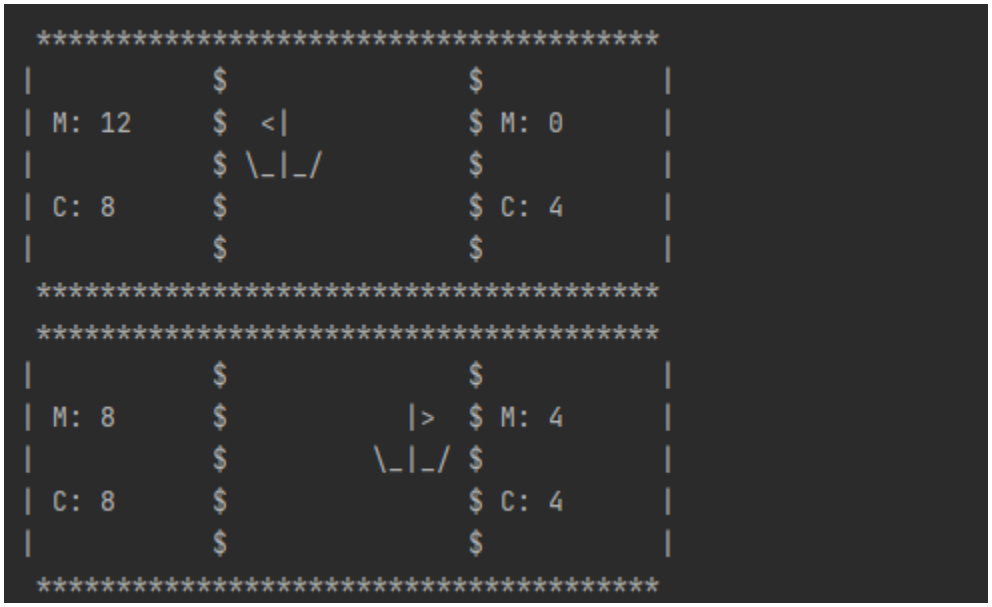

Run:
Main
| M: 31 $ <| $ M: 0 |
| $ _|_/ $ |
| C: 31 $ $ C: 0 |
| M: 31 $ |> $ M: 0 |
| $ _|_/ $ |
| C: 27 $ $ C: 4 |
| M: 31 $ <| $ M: 0 |
| $ _|_/ $ |
| C: 28 $ $ C: 3 |
| M: 31 $ |> $ M: 0 |
| $ _|_/ $ |
| C: 25 $ $ C: 6 |
| M: 31 $ <| $ M: 0 |
| $ _|_/ $ |
| C: 27 $ $ C: 4 |
| M: 27 $ |> $ M: 4 |
| $ _|_/ $ |
| C: 27 $ $ C: 4 |

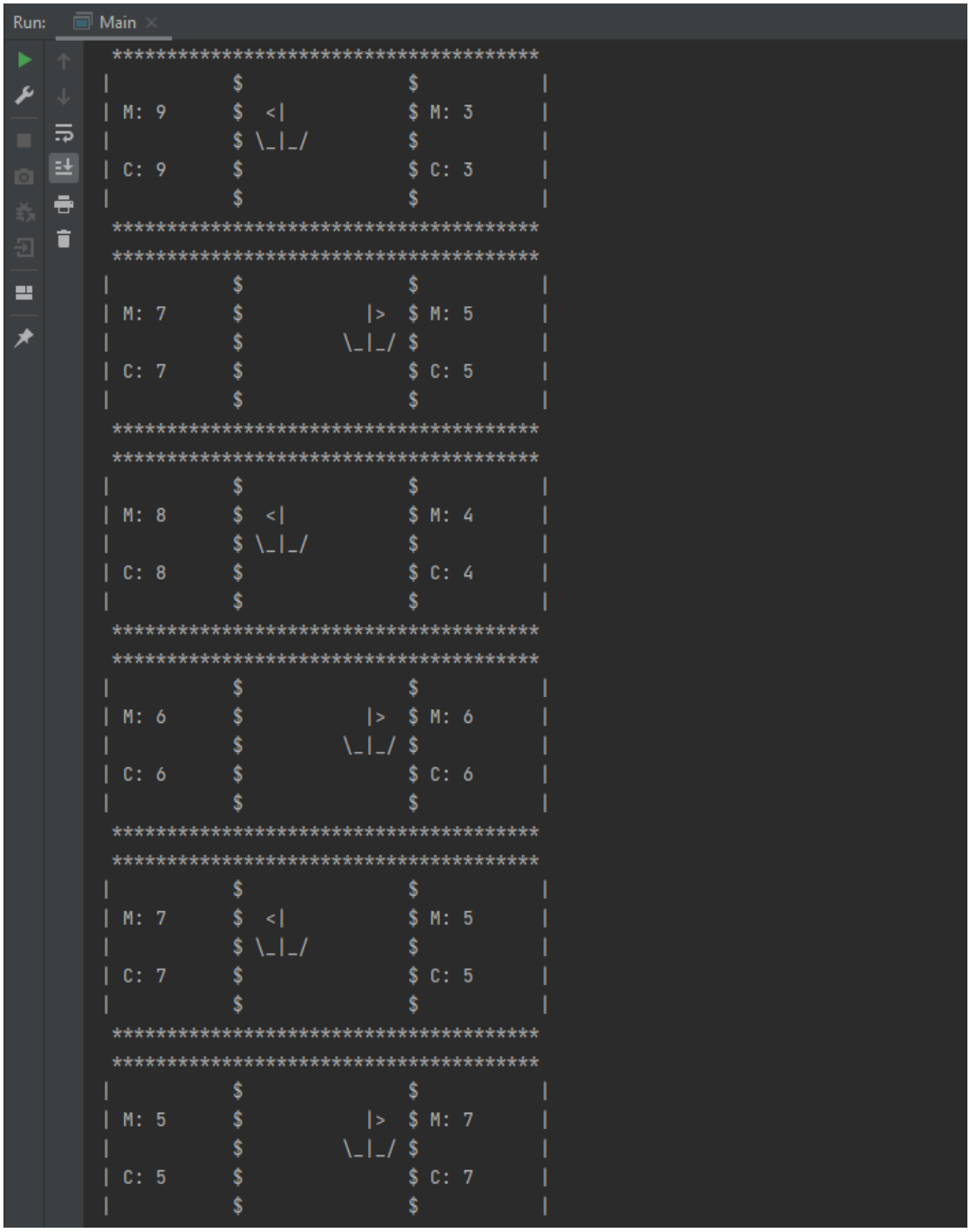
...


| M: 0 $ |> $ M: 31 |
| $ _|_/ $ |
| C: 0 $ $ C: 31 |
Search time: 0.008 seconds
Final Crossings: 59
Process finished with exit code 0

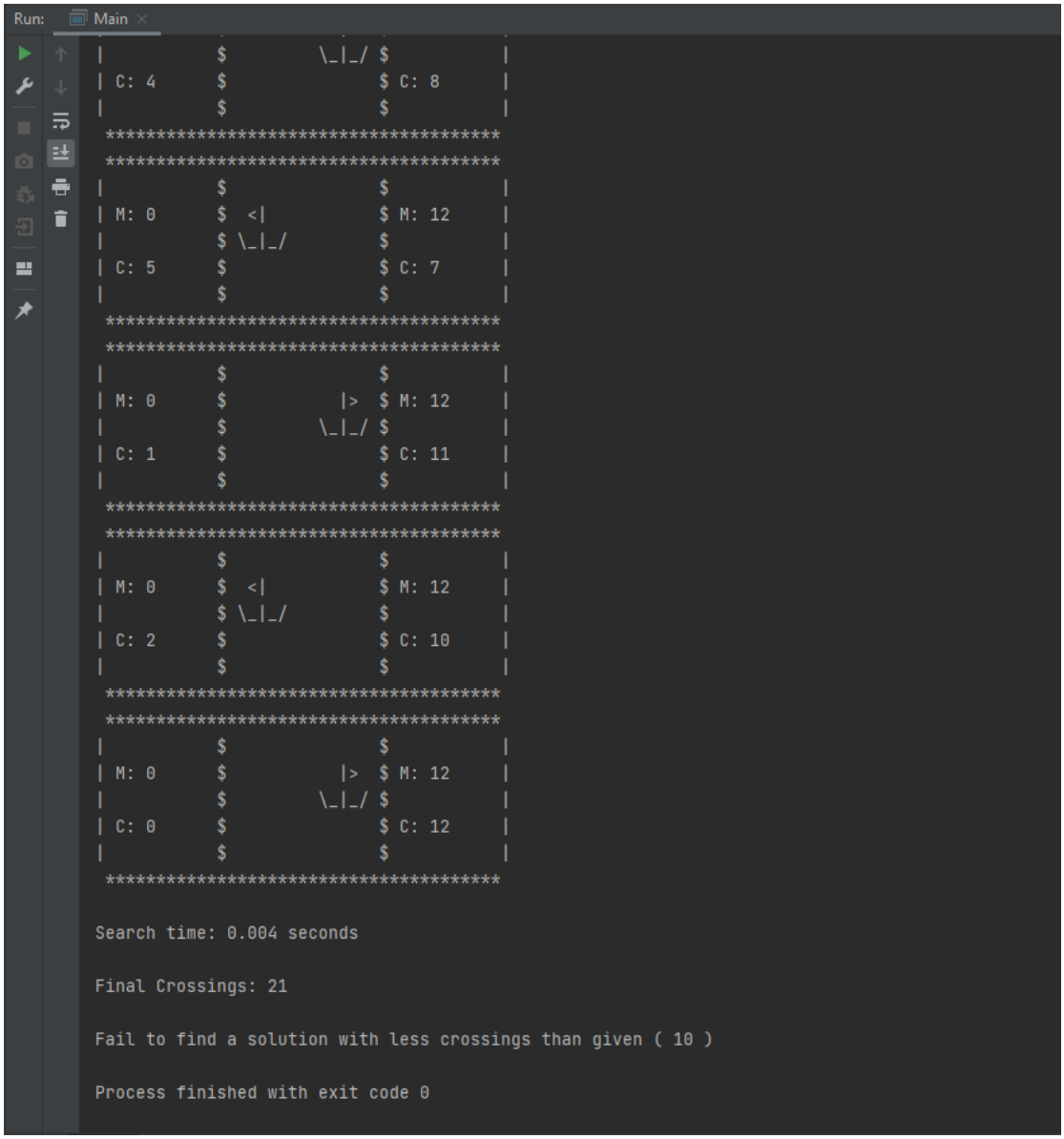
5) N = 12, M = 4, K = 10


Run:
Main
"C:\Program Files\Java\jdk1.8.0_241\bin\java.exe" ...
Give the number of Missionaries and Cannibals on the left side:
12
Give the capacity of the boat:
4
Give the maximum number of crossings allowed:
10
********** MISSIONARIES AND CANNIBALS **********
Missionaries: 12, Cannibals: 12, Boat capacity: 4, Maximum number of crossings allowed: 10
********** START **********
| $ $ |
| M: 12 $ <| $ M: 0 |
| $ _|_/ $ |
| C: 12 $ $ C: 0 |
| $ $ |
| $ $ |
| M: 12 $ |> $ M: 0 |
| $ _|_/ $ |
| C: 8 $ $ C: 4 |
| $ $ |
| $ $ |
| M: 12 $ <| $ M: 0 |
| $ _|_/ $ |
| C: 9 $ $ C: 3 |
| $ $ |
| $ $ |
| M: 12 $ |> $ M: 0 |
| $ _|_/ $ |
| C: 5 $ $ C: 7 |
| $ $ |


| $ $ |
| M: 12 $ <| $ M: 0 |
| $ _|_/ $ |
| C: 8 $ $ C: 4 |
| $ $ |
| $ $ |
| M: 8 $ |> $ M: 4 |
| $ _|_/ $ |
| C: 8 $ $ C: 4 |
| $ $ |


Run:
Main
M: 9
C: 9
M: 3
C: 3
M: 7
C: 7
M: 5
C: 5
M: 8
C: 8
M: 4
C: 4
M: 6
C: 6
M: 7
C: 7
M: 5
C: 5
M: 5
C: 5
M: 7
C: 7

...


Run:
Main
| $ _|_/ $ |
| C: 4 $ $ C: 8 |
| $ $ |
| M: 0 $ <| $ M: 12 |
| $ _|_/ $ |
| C: 5 $ $ C: 7 |
| M: 0 $ |> $ M: 12 |
| $ _|_/ $ |
| C: 1 $ $ C: 11 |
| M: 0 $ <| $ M: 12 |
| $ _|_/ $ |
| C: 2 $ $ C: 10 |
| M: 0 $ |> $ M: 12 |
| $ _|_/ $ |
| C: 0 $ $ C: 12 |
Search time: 0.004 seconds
Final Crossings: 21
Fail to find a solution with less crossings than given (10)
Process finished with exit code 0