ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
Τμήμα Πληροφορικής

Εργασία Μαθήματος «Τεχνητή Νοημοσύνη και Έμπειρα Συστήματα»

	Προαιρετική Εργασία
Όνομα φοιτητή – Αρ. Μητρώου	Σμύρης Βασίλειος - Π16131
Ημερομηνία παράδοσης	31 - 5 - 2019

Εκφώνηση της άσκησης

2019 – Θέμα προαιρετικής εργασίας για το μάθημα «Τεχνητή Νοημοσύνη και Έμπειρα Συστήματα». Η εργασία είναι ατομική. Ημερομηνία παράδοσης είναι η 31η Μαίου 2019.

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. **Ημερομηνία παράδοσης είναι η 31η Μαίου 2019.**

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

1 Εισαγωγή

Η εργασία εκπονήθηκε σε γλώσσα προγραμματισμού C++, στο προγραμματιστικό περιβάλλον CLion. Ο τυφλός αλγόριθμος αναζήτησης που χρησιμοποιήθηκε είναι ο Depth First Search και ο ευριστικός αλγόριθμος που χρησιμοποιήθηκε είναι ο Best-First Search με μια συνάρτηση απόστασης Manhattan ως ευριστική συνάρτηση. Οι αλγόριθμοι μπορούν να εκτελεστούν από ένα πρόγραμμα, το οποίο παραπέμπει το χρήστη να επιλέξει να ποιόν από τους δύο αλγορίθμους θέλει να εκτελέσει. Το πρόγραμμα συμπεριλαμβάνεται στο αρχείο της εργασίας.

2 Ανάλυση προγράμματος

Παρακάτω ακολουθεί μια παρουσίαση του κώδικα που πραγματοποιεί την επίλυση του προβλήματος και της λογικής που ακολουθεί. Η σειρά με την οποία θα αναλυθούν τα διάφορα μέρη του προγράμματος ακολουθεί τη σειρά με την οποία κάθε μέρος θα χρησιμοποιόταν σε ένα ρεαλιστικό σενάριο εκτέλεσης του προγράμματος.

2.1 Έναρξη

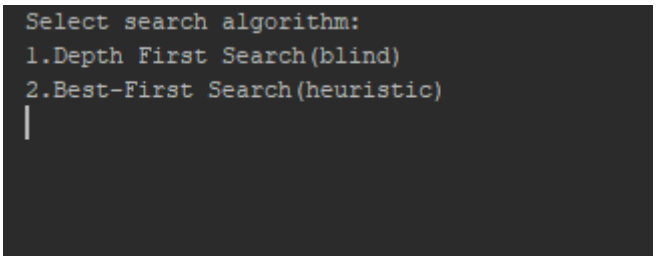
Σε αυτή την υποενότητα θα παρουσιαστεί ο κώδικας που καλωσορίζει το χρήστη στο πρόγραμμα και ξεκινά τη λύση του προβλήματος. Στην παρακάτω εικόνα φαίνεται ο κώδικας που κάνει τη δουλειά αυτή.

Οι γραμμές 209 έως 219, ενημερώνουν το χρήστη για τις επιλογές που έχει μέσω ενός μηνύματος στην κονσόλα, και τον παραπέμπουν να επιλέξει μία από αυτές. Ο χρήστης επιλέγει εισάγοντας τους αριθμούς 1 ή 2 και πατώντας enter. Η while loop που φαίνεται στην εικόνα, σιγουρεύει ότι ο χρήστης μπορεί να ξαναδώσει μία είσοδο εάν βάλει κάποια λανθασμένη. Ανάλογα με την είσοδο, το πρόγραμμα εκτελεί με όρισμα την αρχική κατάσταση είτε τον τυφλό, είτε τον ευριστικό αλγόριθμο.

Η καταστάσεις του προβλήματος στο πρόγραμμα, αναπαριστώνται ως πίνακες τριών ακεραίων. Ο πρώτος αριθμός, αντιπροσωπεύει τον αριθμό των ιεραποστόλων που βρίσκονται στη λάθος όχθη του ποταμού, την αριστερή σε αυτή την περίπτωση. Ο δεύτερος αριθμός, αντιπροσωπεύει τον αριθμό των κανίβαλων που βρίσκονται στη λάθος όχθη. Ο τρίτος αριθμός δείχνει σε ποια όχθη βρίσκεται η βάρκα. Όταν είναι ένα, σημαίνει ότι η βάρκα βρίσκεται στη αριστερή όχθη, ενώ όταν είναι μηδέν, σημαίνει ότι βρίσκεται στη δεξιά, τη σωστή όχθη. Στη γραμμή 206, ορίζεται η αρχική κατάσταση του προβλήματος, ως ένας πίνακας με τους αριθμούς 3, 3 και 1, μιας και στο πρόβλημα υπάρχουν τρεις ιεραπόστολοι, τρεις κανίβαλοι και μια βάρκα, που στην αρχή όλη βρίσκονται στη αριστερή όχθη.

```
205  int main() {
206      array<int, 3> state = {3, 3, 1};
207      int selection;
208      bool f;
209      cout << "Select search algorithm:" << endl << "1.Depth First Search(blind)" << endl << "2.Best-First Search(heuristic)" << endl;
210      cin >> selection;
211      while(selection != 1 && selection != 2) {
212          cin >> selection;
213      }
214      if (selection == 1){
215          f = blindSearch(state);
216      }
217      else{
218          f = heuristicSearch(state);
219      }
```

Εικόνα 1 - Πρώτο μισό της συνάρτησης main. Αναλαμβάνει την αλληλεπίδραση του προγράμματος με το χρήστη και την αρχικοποίηση του προβλήματος.



Παράδειγμα 1 - Στιγμιότυπο από την εκτέλεση του προγράμματος. Εδώ φαίνεται το μενού που καλωσορίζει το χρήστη.

Στην Εικόνα 2, φαίνονται οι βιβλιοθήκες που χρησιμοποιεί το πρόγραμμα, και δύο δομές δεδομένων οι οποίες βρίσκονται στο global scope του προγράμματος, καθώς είναι σκόπιμο να είναι προσβάσιμες από πολλές συναρτήσεις. Αυτές οι δομές είναι τύπου vector, ένας τύπος ο οποίος μοιάζει με την default array της C++, αλλά είναι ποιο αποδοτικός και μπορεί να αλλάζει εύκολα μέγεθος κατά την εκτέλεση του προγράμματος, πράγμα χρήσιμο για τα δεδομένα τα οποία αντιπροσωπεύουν, καθώς δεν ξέρουμε το πλήθος τους εξ αρχής. Το vector cs, αντιπροσωπεύει το Κλειστό Σύνολο (Closed Set) του προβλήματος, τις καταστάσεις που έχουν εξεταστεί. Στο vector step αποθηκεύονται οι καταστάσεις που αποτελούν το μονοπάτι της λύσης του προβλήματος, αν αυτή υπάρχει ή έχει βρεθεί.

```
1  #include <iostream>
2  #include <vector>
3  #include <array>
4  #include <cstdlib>
5
6  using namespace std;
7
8  vector<int> cs;
9  vector<int> steps;
```

Εικόνα 2 - Βιβλιοθήκες και δομές κοινής χρήσης

2.2 Τυφλός Αλγόριθμος

Σε αυτή την υποενότητα, θα παρουσιαστούν τα κομμάτια του κώδικα που συνεργάζονται για να πραγματοποιηθεί η λύση του προβλήματος με έναν τυφλό αλγόριθμο αναζήτησης και συγκεκριμένα τον DFS. Η συνάρτηση που αποτελεί τον κύριο κορμό του αλγορίθμου αυτού φαίνεται στην Εικόνα 3.

Η λογική που ακολουθείται είναι η εξής. Η συνάρτηση είναι τύπου bool και επιστρέφει true αν βρει κάποια λύση και false αν δε βρει. Δέχεται ως παράμετρο έναν πίνακα τριών ακεραίων, ο οποίος αντιπροσωπεύει την τρέχουσα κατάσταση του προβλήματος. Η συνάρτηση αυτή εφαρμόζεται σε κάθε κατάσταση του προβλήματος αναδρομικά, δηλαδή για κάθε κατάσταση για την οποία καλείται αυτή η συνάρτηση, καλεί τον εαυτό της για τα παιδιά της τρέχουσας κατάστασης και μετά για τα παιδιά των παιδιών της κ.ο.κ. Αν κάποιο παιδί σε αυτό το υποθετικό δέντρο καταστάσεων που δημιουργείται είναι η τελική κατάσταση, επιστρέφεται true, το οποίο "αντηχεί" σε όλες τις συναρτήσεις αναζήτησης που βρίσκονται σε εξέλιξη, από την τρέχουσα, μέχρι την πηγαία συνάρτηση, ή αλλιώς τη ρίζα του δέντρου

καταστάσεων, η οποία με τη σειρά της επιστρέφει αυτό το true στην κύρια συνάρτηση. Αν δε βρεθεί κάποια λύση, θα γίνει το ίδιο μέ ένα σήμα false, αφού έχουν εξεταστεί όλοι οι πιθανοί κόμβοι του δέντρου.

```
81  bool blindSearch(array<int, 3> state){
82      bool f;
83      if(!checkClosedSet(state)){
84          if(!isFinal(state)){
85              if(isValid(state)){
86                  if(state[2] == 1){
87                      f = blindSearch({state[0] - 1, state[1] - 1, 0});
88                      if(!f) {
89                          f = blindSearch({state[0] - 2, state[1], 0});
90                          if (!f) {
91                              f = blindSearch({state[0], state[1] - 2, 0});
92                              if (!f) {
93                                  f = blindSearch({state[0] - 1, state[1], 0});
94                                  if(!f){
95                                      f = blindSearch({state[0], state[1] - 1, 0});
96                                  }
97                              }
98                          }
99                      }
100                 }
101                 else{
102                     f = blindSearch({state[0] + 1, state[1] + 1, 1});
103                     if(!f) {
104                         f = blindSearch({state[0] + 2, state[1], 1});
105                         if (!f) {
106                             f = blindSearch({state[0], state[1] + 2, 1});
107                             if (!f) {
108                                 f = blindSearch({state[0] + 1, state[1], 1});
109                                 if(!f){
110                                     f = blindSearch({state[0], state[1] + 1, 1});
111                                 }
112                             }
113                         }
114                     }
115                 }
116                 if(f){
117                     addToSteps(state);
118                 }
119                 return f;
120             }
121             return false;
122         }
123         addToSteps(state);
124         return true;
125     }
126     return false;
127 }
```

Εικόνα 3 - Κύριος κορμός του τυφλού αλγορίθμου αναζήτησης.

Παρατηρώντας λεπτομερέστερα τον κώδικα της Εικόνας 3, βλέπουμε ότι αρχικά (γραμμή 83) η τρέχουσα κατάσταση ελέγχεται για τον αν βρίσκεται στο Κλειστό Σύνολο(ΚΣ). Αν βρίσκεται, τότε η συνάρτηση αναζήτησης επιστρέφει false, αλλιώς συνεχίζεται ο έλεγχος της κατάστασης. Ο έλεγχος για το αν υπάρχει η κατάσταση στο ΚΣ γίνεται με τη συνάρτηση checkClosedSet, ο κώδικας της οποίας φαίνεται στην Εικόνα 4. Η συνάρτηση αυτή παίρνει ως είσοδο την τρέχουσα κατάσταση. Αν το ΚΣ δεν είναι άδειο, δηλαδή αν υπάρχουν δεδομένα στο ΚΣ για να τα συγκρίνουμε με την τρέχουσα κατάσταση, τότε τη συγκρίνουμε με όλες τις καταστάσεις του ΚΣ. Η σύγκριση γίνεται σε τρία στάδια, ένα για κάθε συνιστώσα της κατάστασης. Αν η σύγκριση δύο όμοιων συνιστωσών 2 καταστάσεων δείξει ότι δεν είναι ίδιες, τότε η σύγκριση διακόπτεται. Αν επιτύχουν όλες οι συγκρίσεις, τότε η συνάρτηση επιστρέφει true για να δηλώσει ότι η τρέχουσα κατάσταση υπάρχει στο ΚΣ. Αν το ΚΣ είναι άδειο, δηλαδή αν βρισκόμαστε στην αρχική κατάσταση, ή αν η τρέχουσα κατάσταση δεν υπάρχει σε αυτό, τότε η κατάσταση προστίθεται στο ΚΣ και επιστρέφεται false για να δηλωθεί ότι δεν βρέθηκε η κατάσταση στο ΚΣ. Κάτι που πρέπει να σημειωθεί σε αυτό το σημείο, είναι ότι καταστάσεις στα vectors δεν αποθηκεύονται ως πίνακες τριών τιμών, αλλά οι συνιστώσες τους αποθηκεύονται ως ξεχωριστές, αλλά διαδοχικές τιμές. Γι' αυτό και το βήμα του for loop είναι 3. Το i αντιπροσωπεύει την πρώτη συνιστώσα μιας κατάστασης και υπόλοιπες συνιστώσες γίνονται προσβάσιμές με την προσθήκη ενός offset στο i.

```
36  bool checkClosedSet(array<int, 3> state){
37      if(!cs.empty()) {
38          for(unsigned int i = 0; i < cs.size(); i += 3){
39              if(cs[i] == state[0]){
40                  if(cs[i + 1] == state[1]){
41                      if(cs[i + 2] == state[2]){
42                          return true;
43                      }
44                  }
45              }
46          }
47      }
48      cs.push_back(state[0]);
49      cs.push_back(state[1]);
50      cs.push_back(state[2]);
51      return false;
52  }
```

Εικόνα 4 - Συνάρτηση checkClosedSet. Αυτή η συνάρτηση ελέγχει αν μια κατάσταση βρίσκεται στο κλειστό σύνολο.

Έπειτα από αυτό τον έλεγχο, ελέγχεται αν η τρέχουσα κατάσταση είναι τελική (Εικόνα 3 - γραμμή 84). Αν η κατάσταση δεν είναι τελική, συνεχίζεται η επεξεργασία της, αλλιώς, η κατάσταση προστίθεται στο vector με τα βήματα του μονοπατιού της λύσης και επιστρέφεται true, το οποίο θα μεταφερθεί μέχρι την πηγαία συνάρτηση αναζήτησης και θα οδηγήσει στην εμφάνιση της λύσης και τον τερματισμό του προγράμματος. Η συνάρτηση isFinal, η οποία ελέγχει αν η τρέχουσα κατάσταση είναι τελική, φαίνεται στην Εικόνα 5. Γνωρίζουμε ότι θέλουμε να φτάσουμε σε μια κατάσταση στην οποία όλοι οι ιεραπόστολοι, όλοι οι κανίβαλοι και οι βάρκα βρίσκονται στη δεξιά όχθη. Επομένως, η τελική κατάσταση αντιπροσωπεύεται

από έναν πίνακα τριών μηδενικών τιμών. Η συνάρτηση isFinal παίρνει τη τρέχουσα κατάσταση και εξετάζει αν όλες της οι συνιστώσες είναι μηδέν. Αν κατά τον έλεγχο, οποιαδήποτε τιμή βρεθεί ότι είναι μη μηδενική, ο έλεγχος διακόπτεται και επιστρέφεται false, αλλιώς επιστρέφεται true για να δηλωθεί ότι η τρέχουσα κατάσταση είναι τελική.

```
18  bool isFinal(array<int, 3> state){
19      if(state[0] == 0){
20          if(state[1] == 0){
21              if(state[2] == 0){
22                  return true;
23              }
24          }
25      }
26      return false;
27  }
```

Εικόνα 5 - Συνάρτηση isFinal. Ελέγχει εάν μια κατάσταση είναι τελική.

Μετά από αυτό τον έλεγχο, εκτελείται έναν τελευταίος έλεγχος στην κατάσταση, ο οποίος δεν είναι μέρος του αλγορίθμου DFS, αλλά έρχεται να ανταποκριθεί στους περιορισμούς του προβλήματος (Εικόνα 3 - γραμμή 85). Η συνάρτηση isValid, έρχεται να εξετάσει αν μια κατάσταση του προβλήματος είναι έγκυρη με βάση τα κριτήρια που αυτό θέτει. Η συνάρτηση φαίνεται στην Εικόνα 6. Υπάρχουν τέσσερις περιπτώσεις όπου μια κατάσταση δεν είναι έγκυρη. Οι δύο περιπτώσεις πηγάζουν από τους κανόνες του προβλήματος. Το πρόβλημα απαιτεί να μην υπάρχουν σε καμία περίπτωση περισσότεροι κανίβαλοι απ' ότι ιεραπόστολοι σε οποιαδήποτε όχθη, καθώς έτσι οι κανίβαλοι θα φάνε τους ιεραπόστολους. Η μία περίπτωση είναι να υπάρχουν περισσότεροι κανίβαλοι απ' ότι ιεραπόστολοι στη αριστερή όχθη, με εξαίρεση την περίπτωση που δεν υπάρχουν ιεραπόστολοι στη αριστερή όχθη, αλλά μόνο κανίβαλοι, καθώς, παρ' όλο που οι κανίβαοι είναι περισσότεροι από τους μηδενικούς ιεραποστόλους, δεν υπάρχουν στην ίδια όχθη ιεραπόστολοι για να φάνε. Η δεύτερη περίπτωση, είναι να υπάρχουν περισσότεροι κανίβαλοι απ' ότι ιεραπόστολοι στη δεξιά όχθη. Καθώς δεν αντιπροσωπεύουμε κάπως την κατάσταση στη δεξιά όχθη, πρέπει να βρούμε μια κατάσταση ή καταστάσεις οι οποίες υπονοούν αυτή την περίπτωση. Έτσι, λοιπόν, μπορούμε να πούμε ότι όταν υπάρχουν κάτω από τρεις ιεραπόστολοι στην αριστερή όχθη και είναι περισσότεροι από τους κανίβαλους, μια τέτοια κατάσταση δεν είναι έγκυρη, καθώς υπονοεί ότι στη άλλη όχθη υπάρχουν τουλάχιστον 1 ιεραπόστολος και 2 κανίβαλοι. Οι άλλες δύο περιπτώσεις στις οποίες μια κατάσταση δεν είναι έγκυρη, είναι όταν είτε ο αριθμός των ιεραποστόλων, είτε των κανιβάλλων είναι μικρότερος του 0 ή μεγαλύτερος του 3. Τέτοιες καταστάσεις μπορούν να προκύψουν λόγω του γενικού τρόπου με το οποίο εφαρμόζονται οι τελεστές μετάβασης, και είναι άκυρες καθώς δε μπορεί ένας αριθμός ατόμων να είναι αρνητικός και δεν έχουμε αρκετούς κανίβαλους και ιεραποστόλους για να μπορεί να είναι μεγαλύτερος του 3.

```
11  bool isValid(array<int, 3> state){
12      if((state[0] < 0 || state[0] > 3) || (state[1] < 0 || state[1] > 3) || (state[0] > 0 && (state[0] < state[1])) || (state[0] < 3 && (state[0] > state[1]))){
13          return false;
14      }
15      return true;
16  }
```

Εικόνα 6 - Συνάρτηση isValid. Αυτή η συνάρτηση ελέγχει αν μια κατάσταση του προβλήματος είναι έγκυρη με βάση τους περιορισμούς αυτού.

Στη γραμμή 86 (Εικόνα 3), μπαίνουμε στο στάδιο της εφαρμογής τελεστών μετάβασης στην τρέχουσα κατάστασή για την εύρεση των παιδιών της. Οι τελεστές μετάβασης σε αυτή την περίπτωση είναι όλοι οι πιθανοί συνδυασμοί ατόμων που μπορούμε να μεταφέρουμε κάθε φορά από την όχθη που βρίσκεται η βάρκα στην άλλη. Η βάρκα χρειάζεται τουλάχιστον ένα άτομο για να κινηθεί και μπορεί να πάρει μέχρι δύο άτομα. Επομένως, μπορούμε να μετακινήσουμε έναν ιεραπόστολο κι έναν κανίβαλο, 2 ιεραποστόλους, 2 κανίβαλους, έναν ιεραπόστολο ή έναν κανίβαλο. Άρα η κάθε κατάσταση έχει πέντε παιδιά. Πριν εφαρμοστούν οι τελεστές μετάβασης, εκτελείται ένας έλεγχος ο οποίος βλέπει αν η τιμή της βάρκας της τρέχουσας κατάστασης είναι 1, δηλαδή αν η βάρκα βρίσκεται στην αριστερή όχθη. Αν αυτή η τιμή είναι 1, τότε εφαρμόζονται οι τελεστές μετάβασης, και δημιουργούνται παιδιά των οποίων οι τιμές των ιεραποστόλων και των κανίβαλων είναι μειωμένες κατά τον αριθμό των ατόμων που πέρασαν στην απέναντι όχθη και η τιμή της βάρκας τους γίνεται μηδέν για να αντικατοπτριστεί το γεγονός ότι η βάρκα βρίσκεται πλέον στη δεξιά όχθη. Αν η τιμή της βάρκας της τρέχουσας κατάστασης είναι 0, τότε εφαρμόζονται οι ίδιοι τελεστές, αλλά οι τιμές των παιδιών είναι αυξημένες κατά τον αριθμό των ατόμων που διέσχισαν το ποτάμι. Αυτό μπορεί να δημιουργήσει και μερικές καταστάσεις των οποίων οι τιμές μπορεί να είναι αρνητικές ή μεγαλύτερες του 3, πράγμα που τις κάνει άκυρες, όπως έχει είδη αναφερθεί. Σε αυτό το πρόγραμμα δεν υπάρχει κάποια δομή που αντιπροσωπεύει μέτωπο αναζήτησης. Το μέτωπο αναζήτησης είναι οι καταστάσεις που εκκρεμούν να ελεγχθούν σε κάθε συνάρτηση αναζήτησης. Για κάθε κατάσταση, ελέγχεται το πρώτο παιδί της, και έπειτα το πρώτο παιδί του παιδιού της κ.ο.κ. Η συνάρτηση αναζήτησης για κάθε παιδί της κατάστασης επιστρέφει το αποτέλεσμά της στη bool μεταβλητή f. Αν το παιδί επιστρέψει true, τότε η τρέχουσα κατάσταση αποθηκεύεται στο vector steps, το οποίο περιέχει το μονοπάτι της λύσης, αλλιώς καλείται η συνάρτηση αναζήτησης για το επόμενο παιδί της. Αν κανένα παιδί δεν επιστρέψει true, τότε η συνάρτηση της τρέχουσας κατάστασης επιστρέφει false.

2.3 Ευριστικός Αλγόριθμος

Σε αυτή τη υποενότητα θα αναλυθεί ο ευριστικός αλγόριθμος αναζήτησης που μπορεί να εκτελεστεί σε αυτό το πρόγραμμα, του οποίου ο κύριος κορμός φαίνεται στην Εικόνα 7. Ο ευριστικός αλγόριθμος που χρησιμοποιεί το πρόγραμμα είναι ο BFS, και ακολουθεί, όπως και ο προηγούμενος αλγόριθμος, μια DFS λογική, αλλά με το παραπάνω βήμα ότι για κάθε κατάσταση επιλέγει πιο παιδί της θα εξετάσει πρώτα, σύμφωνα με το πιο παιδί έχει τη μικρότερη ευριστική τιμή, η οποία είναι η απόσταση Manhattan του παιδιού από την τελική κατάσταση. Πραγματοποιεί και αυτός έλεγχο ΚΣ, τελικής κατάστασης και εγκυρότητας και καλεί τον εαυτό του αναδρομικά, χρησιμοποιώντας το ίδιο σύστημα επιστροφής μιας τιμής true από την τελική συνάρτηση αναζήτησης, σε όλες τις γονικές συναρτήσεις του, μέχρι την πηγαία συνάρτηση, αν βρεθεί κάποια λύση, αποθηκεύοντας κατά αυτή την επιστροφή κάθε κατάσταση του μονοπατιού της λύσης στο vector steps.

```
144  bool heuristicSearch(array<int, 3> state) {
145      array<array<int, 3>, 5> children;
146      array<int, 5> mv;
147      bool f;
148      if (!checkClosedSet(state)) {
149          if (!isFinal(state)) {
150              if (isValid(state)) {
151                  if (state[2] == 1) {
152                      children[0] = {state[0] - 1, state[1] - 1, 0};
153                      children[1] = {state[0] - 2, state[1], 0};
154                      children[2] = {state[0], state[1] - 2, 0};
155                      children[3] = {state[0] - 1, state[1], 0};
156                      children[4] = {state[0], state[1] - 1, 0};
157                  } else {
158                      children[0] = {state[0] + 1, state[1] + 1, 1};
159                      children[1] = {state[0] + 2, state[1], 1};
160                      children[2] = {state[0], state[1] + 2, 1};
161                      children[3] = {state[0] + 1, state[1], 1};
162                      children[4] = {state[0], state[1] + 1, 1};
163                  }
164                  for (int i = 0; i < children.size(); i++) {
165                      mv[i] = manhattan(children[i][0], children[i][1]);
166                  }
167                  children = sort(children, mv);
168                  f = heuristicSearch(children[0]);
169                  if(!f) {
170                      f = heuristicSearch(children[1]);
171                      if (!f) {
172                          f = heuristicSearch(children[2]);
173                          if (!f) {
174                              f = heuristicSearch(children[3]);
175                              if(!f){
176                                  f = heuristicSearch(children[4]);
177                              }
178                          }
179                      }
180                  }
181                  if(f){
182                      addToSteps(state);
183                  }
184                  return f;
185              }
186              return false;
187          }
188          addToSteps(state);
189          return true;
190      }
191      return false;
192  }
```

Εικόνα 7 - Κώδικας της συνάρτησης heuristicSearch. Αυτός είναι ο κύριος κορμός της ευριστικής αναζήτησης του προγράμματος.

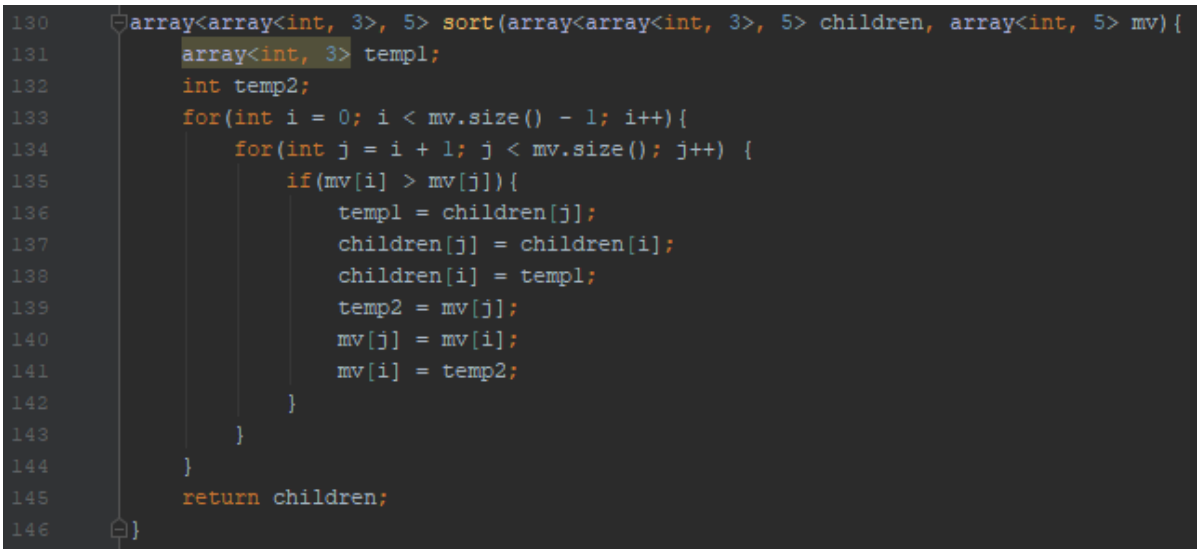
Σε αντίθεση με τον προηγούμενο αλγόριθμο, σε αυτό τον αλγόριθμο πρέπει να επεξεργαζόμαστε τα παιδιά της τρέχουσας κατάστασης πριν τα εξετάζουμε, το οποίο σημαίνει ότι αυτή τη φορά πρέπει να τα αποθηκεύουμε τα παιδιά κάθε κατάστασης κάπου. Για κάθε κατάσταση, λοιπόν, δημιουργείται ένας πίνακας children (Εικόνα 7 - γραμμή 145), ο οποίος κρατά πίνακες τριών ακεραίων και έχει πέντε θέσεις, μία για κάθε παιδί της τρέχουσας κατάστασης. Επίσης, δημιουργείται κι ένας πίνακας mv πέντε ακεραίων, ο οποίος κρατά την ευριστική τιμή κάθε κατάστασης. Στις γραμμές 151 - 163 (Εικόνα 7) γίνεται η εφαρμογή τελεστών μετάβασης στην τρέχουσα κατάσταση, και η αποθήκευση των καταστάσεων παιδιών στον πίνακα children, ανάλογα με τη τιμή της βάρκας της τρέχουσας κατάστασης.

Έπειτα, για κάθε παιδί, εκτελείται η ευριστική συνάρτηση και το αποτέλεσμα της αποθηκεύεται στην κατάλληλη θέση του πίνακα ευριστικών τιμών mv. Η ευριστική συνάρτηση είναι η συνάρτηση manhattan, η οποία φαίνεται στην Εικόνα 8. Αυτή η συνάρτηση βρίσκει την απόσταση μιας κατάστασης από την τελική. Παίρνει ως παραμέτρους τον αριθμό των ιεραποστόλων και τον αριθμό των κανίβαλων μιας κατάστασης. Ο γενικός τύπος της απόστασης Manhattan είναι $|m - m_f| + |c - c_f|$, αλλά μιας και τα m_f και c_f (τα πλήθη ιεραποστόλων και κανίβαλων της τελικής κατάστασης) είναι 0 σε αυτή την περίπτωση, ο τύπος απλοποιείται σε $|m| + |c|$. Οι τελεστές απόλυτης τιμής διατηρούνται, καθώς υπάρχει η πιθανότητα να εχεί η συνάρτηση ως όρισμα κάποιον αρνητικό αριθμό. Η χρήση της απόστασης Manhattan γίνεται με τη λογική ότι όσο λιγότερα άτομα υπάρχουν στη λάθος όχθη, τόσο μικρότερη θα είναι η τιμή.

```
126     int manhattan(int m, int c){
127         return abs(m) + abs(c);
128     }
```

Εικόνα 8 - Συνάρηση manhattan. Βρίσκει την απόσταση Manhattan μιας κατάστασης από την τελική.

Αφού έχουν βρεθεί όλες οι ευριστικές τιμές των παιδιών της τρέχουσας κατάστασης, καλείται η συνάρτηση sort, η οποία ταξινομεί τα παιδιά σύμφωνα με τις ευριστικές τιμές τους, από τη μικρότερη μέχρι τη μεγαλύτερη, και αποθηκεύει το νέο ταξινομημένο πίνακα στον πίνακα children. Ο κώδικας της συνάρτησης sort φαίνεται στην Εικόνα 9. Για την πρώτη μέχρι την τέταρτη θέση του πίνακα, συγκρίνει την τιμή της με τις τιμές των επόμενων θέσεων, και αν η τιμή της είναι μεγαλύτερη, τότε αλλάζει τις τιμές των δύο θέσεων που συγκρίνονται μεταξύ τους, τόσο στις θέσεις του πίνακα ευριστικών τιμών, όσο και στις αντίστοιχες θέσεις του πίνακα παιδιών.



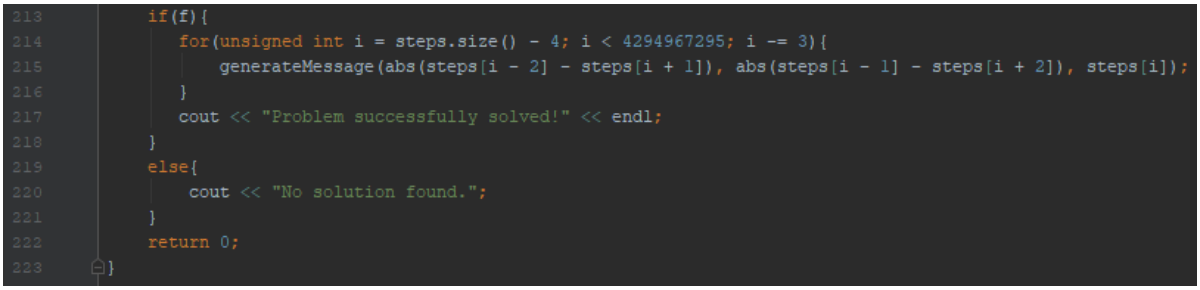
```
130  array<array<int, 3>, 5> sort(array<array<int, 3>, 5> children, array<int, 5> mv){
131      array<int, 3> temp1;
132      int temp2;
133      for(int i = 0; i < mv.size() - 1; i++){
134          for(int j = i + 1; j < mv.size(); j++) {
135              if(mv[i] > mv[j]){
136                  temp1 = children[j];
137                  children[j] = children[i];
138                  children[i] = temp1;
139                  temp2 = mv[j];
140                  mv[j] = mv[i];
141                  mv[i] = temp2;
142              }
143          }
144      }
145      return children;
146  }
```

Εικόνα 9 - Συνάρτηση sort. Τα ταξινομεί έναν πίνακα children σύμφωνα με τις αντίστοιχες τιμές του πίνακα mv, κατά φθίνουσα σειρά.

Μετά την ταξινόμηση, καλείται η συνάρτηση αναζήτησης στα παιδιά της τρέχουσας κατάστασης, με την ίδια DFS λογική όπως και στον προηγούμενο αλγόριθμο.

2.4 Εμφάνιση Λύσης

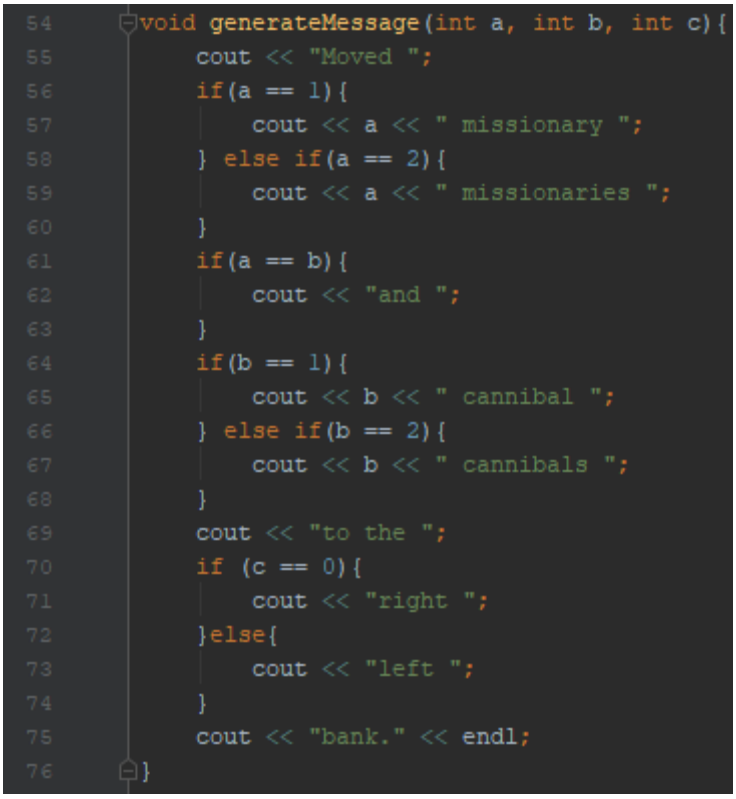
Σε αυτή την υποενότητα, αναλύεται ο κώδικας που αναλαμβάνει την εμφάνιση της λύσης στο χρήστη. Στην Εικόνα 10 φαίνεται το δεύτερο μισό της συνάρτησης main. Ο αλγόριθμος αναζήτησης που είχε επιλέξει ο χρήστης επιστρέφεται στη bool μεταβλητή f. Αν η αναζήτηση επιστρέψει false, το πρόγραμμα εκτυπώνει ένα μήνυμα αποτυχίας λύσης του προγράμματος. Αν επιστρέψει true, το πρόγραμμα εκτυπώνει στη κονσόλα μια σειρά μηνυμάτων που περιγράφουν τα βήματα που ακολούθησε ο αλγόριθμος για να λύσει το πρόγραμμα. Για κάθε κατάσταση στο vector steps, στο vector με τα βήματα της λύσης, εκτυπώνεται ένα κατάλληλο μήνυμα που δημιουργείται από τη συνάρτηση generateMessage, ο κώδικας της οποίας φαίνεται στην Εικόνα 11. Το vector steps διαβάζεται ανάποδα, από την προτελευταία κατάσταση που περιέχει μέχρι την πρώτη. Αυτό γίνεται διότι οι αλγόριθμοι αναζήτησης καταχωρούν τις καταστάσεις ανάποδα, καθώς επιστρέφουν true στην προηγούμενή τους. Ως ορίσματα στη συνάρτηση generateMessage περνιούνται οι απόλυτες τιμές των διαφορών των αριθμών των κανίβαλων και των ιεραποστόλων της τρέχουσας κατάστασης με την προηγούμενή της και ο αριθμός της βάρκας της τρέχουσας κατάστασης.



```
213         if(f){
214             for(unsigned int i = steps.size() - 4; i < 4294967295; i -= 3){
215                 generateMessage(abs(steps[i - 2] - steps[i + 1]), abs(steps[i - 1] - steps[i + 2]), steps[i]);
216             }
217             cout << "Problem successfully solved!" << endl;
218         }
219         else{
220             cout << "No solution found.";
221         }
222         return 0;
223     }
```

Εικόνα 10 - Δεύτερο μισό της συνάρτησης main. Ελέγει το αποτέλεσμα της αναζήτησης και εκτυπώνει κατάλληλο μήνυμα.

Η συνάρτηση generateMessage δημιουργεί δυναμικά μηνύματα που λένε στο χρήστη την κίνηση που έκανε ο αλγόριθμος αναζήτησης. Αρχικά εκτυπώνεται η λέξη "Moved". Αν η διαφορά του αριθμού των ιεραποστόλων της τρέχουσας κατάστασης από αυτόν της προηγούμενης είναι μεγαλύτερη του μηδέν, εκτυπώνεται η διαφορά και ο ενικός ή πληθυντικός της λέξης "missionary". Αν η διαφορά των ιεραποστόλων και των κανίβαλων είναι ίδια, σημαίνει ότι στην τελευταία κίνηση μεταφέρθηκε και ιεραπόστολος και κανίβαλος, οπότε εκτυπώνεται η λέξη "and". Μετά, εκτυπώνεται και η διαφορά των κανίβαλων, αν άλλαξε όχθη κάποιος σε σχέση με την προηγούμενη κατάσταση. Τέλος, εκτυπώνεται η κατεύθυνση προς την οποία έγινε η μεταφορά. Αν η τιμή της βάρκας είναι 1, σημαίνει ότι η κίνηση έγινε προς τα αριστερά, ενώ αν είναι 0 σημαίνει ότι η κίνηση έγινε προς τα δεξιά.



```
54  void generateMessage(int a, int b, int c){
55      cout << "Moved ";
56      if(a == 1){
57          cout << a << " missionary ";
58      } else if(a == 2){
59          cout << a << " missionaries ";
60      }
61      if(a == b){
62          cout << "and ";
63      }
64      if(b == 1){
65          cout << b << " cannibal ";
66      } else if(b == 2){
67          cout << b << " cannibals ";
68      }
69      cout << "to the ";
70      if (c == 0){
71          cout << "right ";
72      }else{
73          cout << "left ";
74      }
75      cout << "bank." << endl;
76  }
```

Εικόνα 11 - Συνάρτηση generateMessage. Δημιουργεί ένα μήνυμα σύμφωνα με τις τιμές των παραμέτρων της.

Ακολουθούν παραδείγματα από την εκτέλεση των δύο αλγορίθμων.

```
Select search algorithm:
1.Depth First Search(blind)
2.Best-First Search(heuristic)
1
Moved 1 missionary and 1 cannibal to the right bank.
Moved 1 missionary to the left bank.
Moved 2 cannibals to the right bank.
Moved 1 cannibal to the left bank.
Moved 2 missionaries to the right bank.
Moved 1 missionary and 1 cannibal to the left bank.
Moved 2 missionaries to the right bank.
Moved 1 cannibal to the left bank.
Moved 2 cannibals to the right bank.
Moved 1 missionary to the left bank.
Moved 1 missionary and 1 cannibal to the right bank.
Problem successfully solved!

Process finished with exit code 0
```

Παράδειγμα 2 - Εκτέλεση του τυφλού DFS αλγορίθμου αναζήτησης.

```
Moved 1 missionary and 1 cannibal to the right bank.
Moved 1 missionary to the left bank.
Moved 2 cannibals to the right bank.
Moved 1 cannibal to the left bank.
Moved 2 missionaries to the right bank.
Moved 1 missionary and 1 cannibal to the left bank.
Moved 2 missionaries to the right bank.
Moved 1 cannibal to the left bank.
Moved 2 cannibals to the right bank.
Moved 1 missionary to the left bank.
Moved 1 missionary and 1 cannibal to the right bank.
Problem successfully solved!

Process finished with exit code 0
```

Παράδειγμα 3 - Εκτέλεση του ευριστικού BFS αλορίθμου αναζήτησης.