1η υποχρεωτική εργασία μαθήματος

Πρόβλημα κανίβαλων και ιεραπόστολων με 2 αλγορίθμους αναζήτησης

Μέλη ομάδας
Φώτης Σταυρόπουλος Π18145
Γιάννης Τουλούπης Π18217

Κλήση αλγορίθμων

```
def main():
    algo=input("Επιλέξτε έναν από τους δύο αλγορίθμους αντίστοιχα:\n"
               "Πληκτρολογώντας '1' και πατώντας Enter γίνεται επιλογή του Breadth - First Search.\n"
               "Πληκτρολογώντας '2' και πατώντας Enter γίνεται επιλογή του Depth - First Search.\n")
    if algo == "1":
        solution = breadth_first_search()
        print("Missionaries and Cannibals AI Problem Solution using Breath - First Search:")
        print_path(solution)
    elif algo == "2":
        solution = depth_first_search()
        print("Missionaries and Cannibals AI Problem Solution using Depth - First Search:")
        print_path(solution)
    else:
        print("Λάθος τιμή εισαγωγής,παρακαλούμε εισάγετε μια αποδεκτή τιμή( '1' ή '2' )")


if __name__ == "__main__":
    main()
```

Αρχικά για την ευκολία του χρήστη έχει δημιουργηθεί ένα μενού που δίνεται η δυνατότητα επιλογής αλγορίθμου που θα λύσει το πρόβλημα, πληκρολογώντας 1 για την επιλογή του BFS και 2 για τον DFS αντίστοιχα.Σε περίπτωση που ο

χρήστης δώσει εσφαλμένη τιμή εισόδου,εμφανίζεται ειδοποίηση σφάλματος στον χρήστη και τερματίζει το πρόγραμμα.

Breadth – First Search (BFS)

Αφού επιλεχθεί ο BFS ορίζουμε ως αρχικές καταστάσεις πρώτα από όλα τα left και right τα οποία συμβολίζουν τις όχθες και το πλήθος ιεραποστολων και κανίβαλων ξεχωριστά. Στην συνέχεια φτιάχνουμε το root_data το οποίο αναφέρεται σε μία κατάσταση η οποία περιέχει τα ορίσματα left και right που αναφέραμε και την θέση του πλοίου για κάθε στιγμή.

Ύστερα δημιουργούμε την λίστα explored η οποία περιέχει τα μονοπάτια που έχουμε διασχίσει, τα nodes που περιέχουν τις καταστάσεις και το path το οποίο είναι το μονοπάτι που έχουμε ακολουθήσει μέχρι την λύση του προβλήματος.

Επιπλέον, προσθέτουμε μία αρχική κατάσταση στα nodes και στην συνέχεια έχοντας μπει στην `while len(nodes) > 0:` αφαιρούμε την πρώτη κατάσταση από τα nodes καθώς χρησιμοποιούμαι αλγόριθμο που λειτουργεί με λογική FIFO και προσθέτουμε το μονοπάτι g στα μέχρι τώρα μονοπάτια που έχουμε επισκεφθεί (explored[]).

Επιπρόσθετα γίνεται έλεγχος για το αν το πλοίο που βρίσκεται στα δεξιά έχει μεταφέρει όλους τους κανίβαλους και ιεραποστόλους στην δεξιά όχθη, έτσι ώστε να τερματίσει ο αλγόριθμος.

Σε περίπτωση που δεν τερματίσει ο αλγόριθμος συνεχίζει ψάχνοντας και επιστρέφοντας μονοπάτια και τα αποκλείει στο ενδεχόμενο που έχουν ήδη επισκεφθεί, αλλιώς προσθέτει το μονοπάτι στο τέλος των nodes.

```
def breadth_first_search():
    left = CoastState(3, 3)
    right = CoastState(0, 0)
    root_data = {"left": left, "right": right, "boat": "left"}

    explored = [] #an diasxisoume ena monopati mpainei sta explored
    nodes = []
    path = []
    nodes.append(GameState(root_data))

    while len(nodes) > 0:
        g = nodes.pop(0)
        explored.append(g)
        if g.data["right"].goal_coast():
            path.append(g)
            return g
        else:
            next_children = g.building_tree() #an uparxei kai alli katastasi
            for x in next_children: # an den uparxei to children pou vrethike prosthese tin
                if (x not in nodes) or (x not in explored):
                    nodes.append(x)
    return None
```

Depth – First Search (DFS)

Αφού επιλεχθεί ο DFS ορίζουμε ως αρχικές καταστάσεις πρώτα από όλα τα left και right τα οποία συμβολίζουν τις όχθες και το πλήθος ιεραποστολων και κανίβαλων ξεχωριστά. Στην συνέχεια φτιάχνουμε το root_data το οποίο αναφέρεται σε μία κατάσταση η οποία περιέχει τα ορίσματα left και right που αναφέραμε και την θέση του πλοίου για κάθε στιγμή.

Ύστερα δημιουργούμε την λίστα explored η οποία περιέχει τα μονοπάτια που έχουμε διασχίσει, τα nodes που περιέχουν τις καταστάσεις και το path το οποίο είναι το μονοπάτι που έχουμε ακολουθήσει μέχρι την λύση του προβλήματος.

Επιπλέον, προσθέτουμε μία αρχική κατάσταση στα nodes και στην συνέχεια έχοντας μπει στην `while len(nodes) >= 0:` αφαιρούμε την τελευταία κατάσταση από τα nodes καθώς χρησιμοποιούμαι αλγόριθμο που λειτουργεί με λογική LIFO και προσθέτουμε το μονοπάτι g στα μέχρι τώρα μονοπάτια που έχουμε επισκεφθεί.

Επιπρόσθετα γίνεται έλεγχος για το αν το πλοίο που βρίσκεται στα δεξιά έχει μεταφέρει όλους τους κανίβαλους και ιεραποστόλους στην δεξιά όχθη, έτσι ώστε να τερματίσει ο αλγόριθμος.

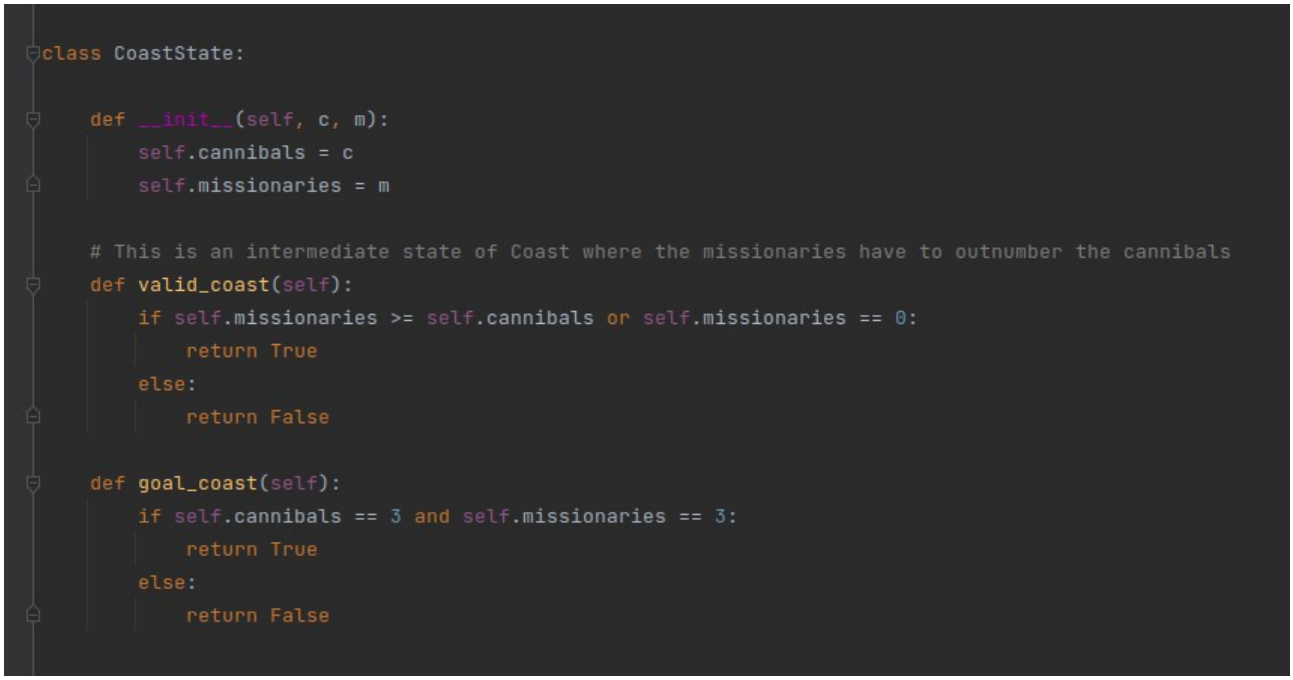
Σε περίπτωση που δεν τερματίσει ο αλγόριθμος συνεχίζει ψάχνοντας και επιστρέφοντας μονοπάτια και τα αποκλείει στο ενδεχόμενο που έχουν ήδη επισκεφθεί, αλλιώς προσθέτει το μονοπάτι στην αρχή των nodes.

```
def depth_first_search():

    left = CoastState(3, 3)
    right = CoastState(0, 0)
    root_data = {"left": left, "right": right, "boat": "left"}

    explored = []
    nodes = []
    path = []
    nodes.append(GameState(root_data))
    while len(nodes) >= 0 :
        g = nodes.pop(-1)
        explored.append(g)
        if g.data["right"].goal_coast():
            path.append(g)
            return g
        else:
            next_children = g.building_tree()
            for x in reversed(next_children):
                if x not in explored :
                    nodes.insert(0,x)
    return None
```

Συνάρτηση valid_coast και goal coast

Η συνάρτηση vaild_coast έχει ως σκοπό τον έλεγχο για το αν είναι επιτρεπτή μία κίνηση, πληρώντας έτσι τους κανόνες για το σύνολο ιεραπόστολων και κανίβαλων για την κάθε όχθη.

Η συνάρτηση goal_coast έχει ως στόχο τον έλεγχο για το αν βρισκόμαστε στην τελική κατάσταση, έτσι ώστε να τερματίσει ο αλγόριθμος.



```python
class CoastState:

    def __init__(self, c, m):
        self.cannibals = c
        self.missionaries = m

    # This is an intermediate state of Coast where the missionaries have to outnumber the cannibals
    def valid_coast(self):
        if self.missionaries >= self.cannibals or self.missionaries == 0:
            return True
        else:
            return False

    def goal_coast(self):
        if self.cannibals == 3 and self.missionaries == 3:
            return True
        else:
            return False
```

Κλάση GameState και συνάρτηση building_tree

Αρχικά ορίζουμε το children = [] όπου θα περιέχει την διαδρομή που έχει δημιουργηθεί.
Επίσης, τα coast και across_coast υπάρχουν για να μπορούμε να δώσουμε την επόμενη κατεύθυνση για το καράβι για την κάθε φορά αναλόγως σε ποια όχθη βρίσκεται την συγκεκριμένη στιγμή.

```
class GameState:

    def __init__(self, data):
        self.data = data
        self.parent = None

    # Creating the Search Tree
    def building_tree(self):
        children = []
        coast = ""
        across_coast = ""
        temp = copy.deepcopy(self.data)
        if self.data["boat"] == "left":
            coast = "left"
            across_coast = "right"
        elif self.data["boat"] == "right":
            coast = "right"
            across_coast = "left"
```

Συνεχίζοντας έχουμε δημιουργήσει ένα μοτίβο περιπτώσεων και κινήσεων μεταφοράς κανιβάλων και ιεραποστόλων για την κάθε όχθη, ενώ ταυτόχρονα γίνεται έλεγχος με την βοήθεια του valid_coast για το αν είναι επιτρεπτή η κίνηση, έτσι ώστε να την προσθέσει στο σύνολο των κινήσεων που δημιουργούν ένα μονοπάτι.

Αρχικά επιχειρούμε την μετακίνηση 2 καννίβαλλων,ύστερα 2 ιεραποστόλων,μετέπειτα 1 ιεραπόστολου ή 1 καννίβαλου και εν τέλει την κοινή μετακίνηση 1 ιεραπόστολου και 1 καννίβαλου. Πριν την προσπάθεια κάθε σεναρίου μέσω της εντολής temp=copy.deepcopy(self.data) ενημερώνουμε τα δεδομένα που διαχειριζόμαστε σε κάθε έλεγχο. Κατά την ολοκλήρωση των ελέγχων η συνάρτηση building_tree(self) επιστρέφει την children[] όπου περιέχει την «διαδρομή» ενεργειών που προήλθε απ'τους ελέγχους.

Missionaries and Cannibals using BFS and DFS algos.py

```
        # MOVING 2 CANNIBALS (CC)
        if temp[coast].cannibals >= 2:
            temp[coast].cannibals = temp[coast].cannibals - 2
            temp[across_coast].cannibals = temp[across_coast].cannibals + 2
            temp["boat"] = across_coast
            if temp[coast].valid_coast() and temp[across_coast].valid_coast():
                child = GameState(temp)
                child.parent = self
                children.append(child)

        temp = copy.deepcopy(self.data)
        # MOVING 2 MISSIONARIES (MM)
        if temp[coast].missionaries >= 2:
            temp[coast].missionaries = temp[coast].missionaries - 2
            temp[across_coast].missionaries = temp[across_coast].missionaries + 2
            temp["boat"] = across_coast
            if temp[coast].valid_coast() and temp[across_coast].valid_coast():
                child = GameState(temp)
                child.parent = self
                children.append(child)

        temp = copy.deepcopy(self.data)
        # MOVING 1 CANNIBAL (C)
        if temp[coast].cannibals >= 1:
            temp[coast].cannibals = temp[coast].cannibals - 1
            temp[across_coast].cannibals = temp[across_coast].cannibals + 1
            temp["boat"] = across_coast
            if temp[coast].valid_coast() and temp[across_coast].valid_coast():
                child = GameState(temp)
                child.parent = self
                children.append(child)

        temp = copy.deepcopy(self.data)
        # MOVING 1 MISSIONARY (M)
        if temp[coast].missionaries >= 1:
            temp[coast].missionaries = temp[coast].missionaries - 1
            temp[across_coast].missionaries = temp[across_coast].missionaries + 1
            temp["boat"] = across_coast
            if temp[coast].valid_coast() and temp[across_coast].valid_coast():
                child = GameState(temp)
                child.parent = self
                children.append(child)

        temp = copy.deepcopy(self.data)
        # MOVING 1 CANNIBAL AND 1 MISSIONARY (CM && MM)
        if temp[coast].missionaries >= 1 and temp[coast].cannibals >= 1:
            temp[coast].missionaries = temp[coast].missionaries - 1
            temp[across_coast].missionaries = temp[across_coast].missionaries + 1
            temp[coast].cannibals = temp[coast].cannibals - 1
            temp[across_coast].cannibals = temp[across_coast].cannibals + 1
            temp["boat"] = across_coast
            if temp[coast].valid_coast() and temp[across_coast].valid_coast():
                child = GameState(temp)
                child.parent = self
                children.append(child)
        return children
```

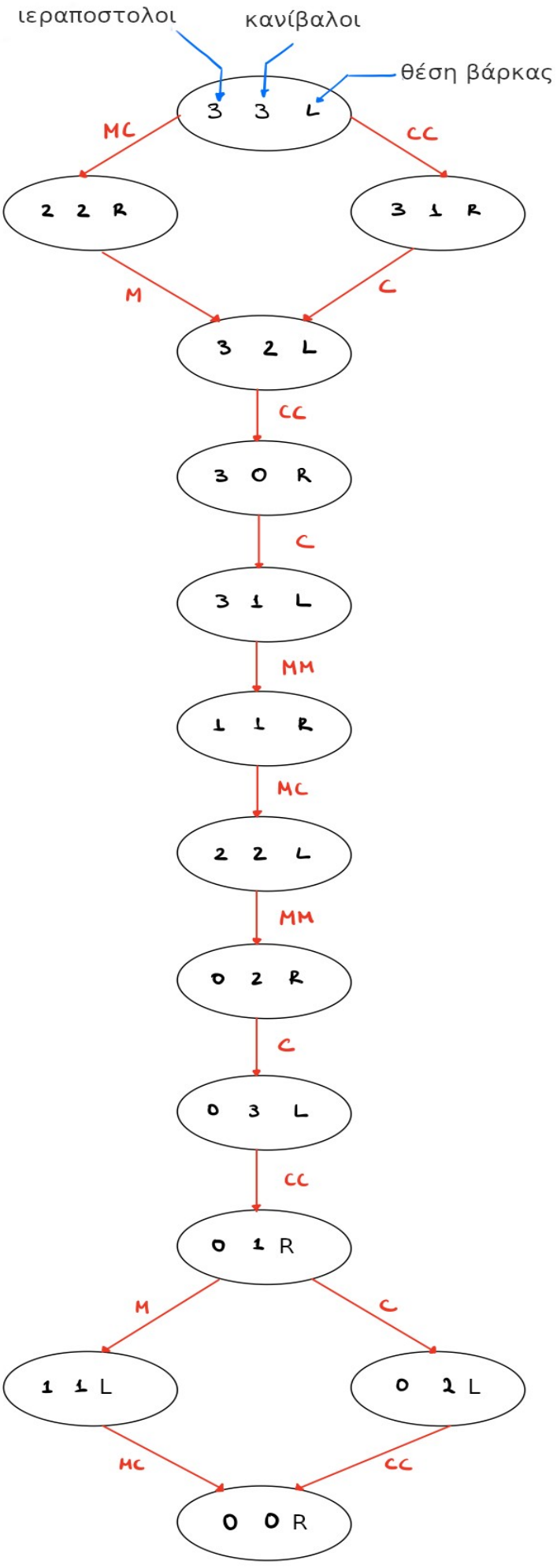
Λύση με BFS

```
Επιλέξτε έναν από τους δύο αλγορίθμους αντίστοιχα:
Πληκτρολογώντας '1' και πατώντας Enter γίνεται επιλογή του Breadth - First Search.
Πληκτρολογώντας '2' και πατώντας Enter γίνεται επιλογή του Depth - First Search.
1
Missionaries and Cannibals AI Problem Solution using Breath - First Search:
                Left Side                    Right Side                  Boat
          Cannibals     Missionaries       Cannibals       Missionaries    Boat Position
State 0    Left C: 3.    Left M: 3.     |   Right C: 0.     Right M: 0.     | Boat: left
State 1    Left C: 1.    Left M: 3.     |   Right C: 2.     Right M: 0.     | Boat: right
State 2    Left C: 2.    Left M: 3.     |   Right C: 1.     Right M: 0.     | Boat: left
State 3    Left C: 0.    Left M: 3.     |   Right C: 3.     Right M: 0.     | Boat: right
State 4    Left C: 1.    Left M: 3.     |   Right C: 2.     Right M: 0.     | Boat: left
State 5    Left C: 1.    Left M: 1.     |   Right C: 2.     Right M: 2.     | Boat: right
State 6    Left C: 2.    Left M: 2.     |   Right C: 1.     Right M: 1.     | Boat: left
State 7    Left C: 2.    Left M: 0.     |   Right C: 1.     Right M: 3.     | Boat: right
State 8    Left C: 3.    Left M: 0.     |   Right C: 0.     Right M: 3.     | Boat: left
State 9    Left C: 1.    Left M: 0.     |   Right C: 2.     Right M: 3.     | Boat: right
State 10    Left C: 2.    Left M: 0.     |   Right C: 1.     Right M: 3.     | Boat: left
State 11    Left C: 0.    Left M: 0.     |   Right C: 3.     Right M: 3.     | Boat: right
End of Path!

Process finished with exit code 0
```

Λύση με DFS

```
Επιλέξτε έναν από τους δύο αλγορίθμους αντίστοιχα:
Πληκτρολογώντας '1' και πατώντας Enter γίνεται επιλογή του Breadth - First Search.
Πληκτρολογώντας '2' και πατώντας Enter γίνεται επιλογή του Depth - First Search.
2
Missionaries and Cannibals AI Problem Solution using Depth - First Search:
                Left Side                    Right Side                  Boat
          Cannibals     Missionaries       Cannibals       Missionaries    Boat Position
State 0    Left C: 3.    Left M: 3.     |   Right C: 0.     Right M: 0.     | Boat: left
State 1    Left C: 2.    Left M: 2.     |   Right C: 1.     Right M: 1.     | Boat: right
State 2    Left C: 2.    Left M: 3.     |   Right C: 1.     Right M: 0.     | Boat: left
State 3    Left C: 0.    Left M: 3.     |   Right C: 3.     Right M: 0.     | Boat: right
State 4    Left C: 1.    Left M: 3.     |   Right C: 2.     Right M: 0.     | Boat: left
State 5    Left C: 1.    Left M: 1.     |   Right C: 2.     Right M: 2.     | Boat: right
State 6    Left C: 2.    Left M: 2.     |   Right C: 1.     Right M: 1.     | Boat: left
State 7    Left C: 2.    Left M: 0.     |   Right C: 1.     Right M: 3.     | Boat: right
State 8    Left C: 3.    Left M: 0.     |   Right C: 0.     Right M: 3.     | Boat: left
State 9    Left C: 1.    Left M: 0.     |   Right C: 2.     Right M: 3.     | Boat: right
State 10    Left C: 1.    Left M: 1.     |   Right C: 2.     Right M: 2.     | Boat: left
State 11    Left C: 0.    Left M: 0.     |   Right C: 3.     Right M: 3.     | Boat: right
End of Path!

Process finished with exit code 0
```

Διάγραμμα


ιεραποστολοι
κανίβαλοι
θέση βάρκας
3 3 L
MC
CC
2 2 R
3 1 R
M
C
3 2 L
CC
3 0 R
C
3 1 L
MM
1 1 R
MC
2 2 L
MM
0 2 R
C
0 3 L
CC
0 1 R
M
C
1 1 L
0 2 L
MC
CC
0 0 R