ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

UNIVERSITY OF PIRAEUS

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

1η ΟΜΑΔΙΚΗ ΕΡΓΑΣΙΑ ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ ΚΑΙ ΕΜΠΕΙΡΑ ΣΥΣΤΗΜΑΤΑ

ΓΚΟΛΕΜΙ ΚΡΙΣΤΙΑΝ, Π18029

ΑΛΕΞΑΝΔΡΟΣ ΓΚΙΝΕΤΣΙ, Π18028

Επίλυση προβλήματος ποτηριών με 2 αλγορίθμους αναζήτησης, γλώσσα προγραμματισμού ελεύθερη.

Παρακάτω παρατίθεται ο κώδικας επίλυσης του προβλήματος, ο οποίος θα αναλυθεί στη συνέχεια:

```
import collections

"""
2 water jugs initially empty: jugA and jugB
jugA's capacity is 70ml and jugB's capacity is 50ml (the jugs do not have markings on them)
The goal is to fill one of the jugs with 40ml of water
The actions we can take are the following:
    1) Fill the first jug
    2) Fill the second jug
    3) Empty the first jug
    4) Empty the second jug
    5) Pour the first jug into the second jug
    6) Pour the second jug into the first jug

visited{}     --> kleisto synolo
path[]        --> mikroskopio
next_states[] --> katastaseis paidia
searchFront   --> metwpo anazhthshs
"""

def main():
    """
    Main function of the code
    """

    initial_state = [[0, 0]]        # initially both jugs are empty
    jugs_capacities = [70, 50]      # list to store the max capacities of the jugs
    goal_amount = 40                # the wanted amount of water in a jug
    visited = {}                    # dictionary to store the visited states
    choice = choose()               # choice calls def choose, if the user chooses DFS choice = True, othe
    search(initial_state, jugs_capacities, goal_amount, visited, choice)


def choose():
    """
    Function that gives the user the ability to choose the algorithm he wants to implement in order to sol
    """
```

```
    press = input("-----CHOOSE THE ALGORITHM YOU WANT TO USE. ENTER 'B' FOR BFS, 'D' FOR DFS: \n")
    # take only the first letter from the user's input
    press = press[0].lower()

    # check for invalid input
    while press != 'b' and press != 'd':
        press = input("INVALID INPUT! CHOOSE THE ALGORITHM YOU WANT TO USE. ENTER 'B' FOR BFS, 'D' FOR DFS
        press = press[0].lower()

    if press == 'd':
        return True
    else:
        return False


def alreadyVisited(state, visited):
    """
    Function to check whether a state has been visited or not
    """

    # if the state has been visited returns True, otherwise returns False
    return visited.get(str(state), False)


def goalReached(path, goal_amount):
    """
    Function to check whether the goal has been reached or no
    """

    if path[-1][0] == goal_amount or path[-1][1] == goal_amount:
        return True
    else:
        return False

def createChildren(jugs_capacities, path, visited):
    """
    Function that creates the possible next states based on the current state
    """

    possible_paths = []     # list that stores the possible paths followed based on the current state
    next_states = []        # list that stores all the possible next states based on the current state
    state = []

    a_max = 70
```

```
b_max = 50

a = path[-1][0]     # initial amount of water in the first jug
b = path[-1][1]     # initial amount of water in the second jug

# 1) fill in the first jug
state.append(a_max)
state.append(b)
if not alreadyVisited(state, visited):
    next_states.append(state)
state = []

# 2) fill in the second jug
state.append(a)
state.append(b_max)
if not alreadyVisited(state, visited):
    next_states.append(state)
state = []

# 3) pour water from the second jug to the first jug
state.append(min(a_max, a + b))
state.append(b - (state[0] - a))
if not alreadyVisited(state, visited):
    next_states.append(state)
state = []

# 4) pour water from the first jug to the second jug
state.append(min(a + b, b_max))
state.insert(0, a - (state[0] - b))
if not alreadyVisited(state, visited):
    next_states.append(state)
state = []

# 5) empty the first jug
state.append(0)
state.append(b)
if not alreadyVisited(state, visited):
    next_states.append(state)
state = []

# 6) empty the second jug
state.append(a)
state.append(0)
if not alreadyVisited(state, visited):
    next_states.append(state)
```

```
    for i in range(0, len(next_states)):
        temp = list(path)
        temp.append(next_states[i])
        possible_paths.append(temp)

    return possible_paths


def search(initial_state, jugs_capacities, goal_amount, visited, choice):
    """
    Function to search using either BFS or DFS for the wanted state and return the followed path
    """

    if choice:
        print("-----IMPLEMENTING DFS\n")
    else:
        print("-----IMPLEMENTING BFS\n")

    found = False

    # search_front represents metwpo anazhthshs
    search_front = collections.deque()
    search_front.appendleft(initial_state)

    while len(search_front) != 0:
        # path represents mikroskopio
        path = search_front.popleft()

        # mark the current state as visited
        visited[str(path[-1])] = True

        if goalReached(path, goal_amount):
            found = True
            goal = path
            break

        next_moves = createChildren(jugs_capacities, path, visited)
        for i in next_moves:
            if choice:
                # implementing DFS
                search_front.appendleft(i)
            else:
                search_front.append(i)
```

```
    if found:
        print("-----THE GOAL HAS BEEN ACHIEVED, PRINTING THE PATH FOLLOWED TO THE GOAL STATE\n")
        for i in range(0, len(goal)):
            print(i, ". ", goal[i])
    else:
        print("-----THE PROBLEM CANNOT BE SOLVED, SORRY\n")


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

Ανάλυση των παραπάνω συναρτήσεων αναλυτικά:

def choose():

Η παραπάνω συνάρτηση δίνει τη δυνατότητα στον χρήστη του προγράμματος να επιλέξει με ποιόν τρόπο θα επιλυθεί το πρόβλημα. Η συνάρτηση δέχεται είσοδο από το πληκτρολόγιο και ανάλογα με το πλήκτρο που θα πατήσει ο χρήστης ορίζεται αν θα χρησιμοποιηθεί BFS ή DFS. Σε περίπτωση που ο χρήστης πατήσει d ή D, η συνάρτηση επιστρέφει True και το πρόβλημα επιλύεται με DFS, ενώ διαφορετικά αν ο χρήστης πατήσει b ή B, η συνάρτηση επιστρέφει False και το πρόβλημα επιλύεται με BFS.

def alreadyVisited(state, visited):

Η συνάρτηση alreadyVisited(), ελέγχει εάν έχουμε επισκεφθεί ήδη μια κατάσταση ή όχι.

```
return visited.get(str(state), False)
```

Μέσω του παραπάνω return statement, η συνάρτηση επιστρέφει True σε περίπτωση που έχουμε επισκεφθεί ήδη την κατάσταση που δίνεται ως όρισμα, ή διαφορετικά False σε περίπτωση που είναι η πρώτη φορά που ελέγχουμε την κατάσταση. Η συνάρτηση get() που καλείται πάνω στο dictionary visited{}, ελέγχει αν υπάρχει το στοιχείο με key str(state) μέσα στο dictionary. Η τρέχουσα κατάσταση state μετατρέπεται σε string μέσω της συνάρτησης str(), καθώς το state είναι λίστα και οι λίστες στην python είναι unhashable, ενώ τα strings είναι hashable.

def goalReached(path, goal_amount):

Η παραπάνω συνάρτηση ελέγχει εάν επιτεύχθηκε ο στόχος του προγράμματος, ο οποίος στη δική μας περίπτωση είναι να φτάσουμε ένα από τα δύο ποτήρια να περιέχουν 40ml νερό.

```
if path[-1][0] == goal_amount or path[-1][1] == goal_amount:
      return True
else:
      return False
```

Η λίστα path περιέχει το μονοπάτι που έχει ακολουθηθεί μέχρι την τρέχουσα κατάσταση, επομένως στο τέλος της λίστας path υπάρχει η τρέχουσα κατάσταση η οποία θα είναι π.χ. της μορφής [[…], […], …, [20, 30]]. Εάν ένα από τα δύο στοιχεία της τελευταίας υπολίστας που είναι και η τρέχουσα κατάσταση είναι ίσο με goal_amount = 40, τότε επετεύχθει ο αρχικός μας στόχος και η συνάρτηση επιστρέφει True, ενώ διαφορετικά επιστρέφει False.

def createChildren(jugs_capacities, path, visited):

Η συνάρτηση createChildren() επιστρέφει μια λίστα από τα πιθανά μονοπάτια που μπορούν να ακολουθηθούν με βάση την τρέχουσα κατάσταση στην οποία βρισκόμαστε. Ανάλογα με την κατάσταση στην οποία βρισκόμαστε, μπορούμε να κάνουμε μια από τις παρακάτω κινήσεις:

1) γέμισε το 1^{ο} ποτήρι

2) γέμισε το 2^{ο} ποτήρι

3) άδειασε το 1^{ο} ποτήρι

4) άδειασε το 2^{ο} ποτήρι

5) ρίξε νερό στο 1^{ο} ποτήρι από το 2^{ο} ποτήρι

6) ρίξε νερό στο 2^{ο} ποτήρι από το 1^{ο} ποτήρι

```
a = path[-1][0]     # initial amount of water in the first jug
b = path[-1][1]     # initial amount of water in the second jug
```

Ορίζονται οι μεταβλητές a, b οι οποίες περιέχουν την τρέχουσα κατάσταση τον δύο ποτηριών τη στιγμή που καλείται η συνάρτηση createChildren(). Το a είναι το 1ο στοιχείο της τελευταίας υπολίστας της λίστας path, ενώ το b είναι αντίστοιχα το 2ο.

Στη συνέχεια δημιουργείται η λίστα next_states[], η οποία περιέχει τις επόμενες καταστάσεις που μπορούμε να μεταβούμε με βάση την τρέχουσα κατάσταση.

```
# 1) fill in the first jug

state.append(a_max)

state.append(b)

if not alreadyVisited(state, visited):

      next_states.append(state)

      state = []
```

Για παράδειγμα μια από τις κινήσεις που μπορούμε να κάνουμε είναι να γεμίσουμε το 1ο ποτήρι. Η λίστα state[] αποθηκέυει μια προσωρινή κατάσταση παιδί, η οποία αν δεν έχει την έχουμε ξαναεπισκεφθεί νωρίτερα αποθηκέυεται στη λίστα next_states[], η οποία περιέχει όλες τις καταστάσεις παιδία της τρέχουσας κατάστασης. Αρχικά, εκτελώντας την κίνηση αυτή η λίστα state[] γίνεται state[a_max, b] που σημαίνει ότι το πρώτο ποτήρι γεμίζει πέρνοντας την τιμή a_max = 70, ενώ το 2ο ποτήρι μένει ως είχε στην προηγούμενη κατάσταση καθώς προς το παρόν δεν μας ενδιαφέρει. Έπειτα γίνεται έλεγχος αν έχουμε επισκεφθεί νωρίτερα τη συγκεκριμένη κατάσταση και αν είναι η πρώτη φορά που την επισκεπτόμαστε, προστίθεται στην λίστα next_states[]. Όμοια λειτουργεί η συνάρτηση και για τις υπόλοιπες κινήσεις που μπορούμε να κάνουμε.

```
for i in range(0, len(next_states)):

      temp = list(path)

      temp.append(next_states[i])

      possible_paths.append(temp)


return possible_paths
```

Τέλος, αφού γίνουν όλες οι κινήσεις που μπορούν να γίνουν με βάση την τρέχουσα κατάσταση, δημιουργείται η λίστα possible_paths η οποία περιέχει όλα τα δυνατά μονοπάτια που μπορούμε να πάρουμε ξεκινώντας από την προηγούμενη κατάσταση, με τον παραπάνω κώδικα. Για παράδειγμα αν η αρχική κατάσταση είναι η [0, 0], η λίστα next_states θα είναι η [[70, 0], [0, 50]] καθώς όταν τα ποτήρια είναι άδεια το μόνο που μπορούμε να κάνουμε είναι να γεμίσουμε ένα από τα δύο ποτήρια. Έχοντας τη λίστα next_states[], δημιουργείται τέλος η λίστα possible_paths[], η οποία στο συγκεκριμένο παράδειγμα θα είναι η εξής: [[[0, 0], [70, 0]], [[0, 0], [0, 50]]].

def search(initial_state, jugs_capacities, goal_amount, visited, choice):

Η συνάρτηση search(), είναι ουσιαστικά η συνάρτηση στην οποία υλοποιείται ο αλγόριθμος BFS ή DFS (ανάλογα με την επιλογή του χρήστη) ώστε να φτάσουμε από την αρχική κατάσταση [[0, 0]] στην επιθυμητή κατάσταση όπου ένα από τα δύο ποτήρια περιέχουν 40 ml νερό.

```
if choice:
      print("-----IMPLEMENTING DFS\n")
else:
      print("-----IMPLEMENTING BFS\n")
```

Ξεκινώντας, αν ο χρήστης έχει πληκτρολογήσει ‘d’, η συνάρτηση choose() επιστρέφει την τιμή True και αφού choice = choose(), η μεταβλητή choice παίρνει και αυτή την τιμή True και η συνάρτηση τυπώνει ‘’-----IMPLEMENTING DFS’’. Όμοια γίνεται στην περίπτωση που ο χρήστης επιλέξει BFS.

```
# search_front represents metwpo anazhthshs
search_front = collections.deque()
search_front.appendleft(initial_state)

while len(search_front) != 0:
      # path represents mikroskopio
      path = search_front.popleft()
```

```
    # mark the current state as visited
    visited[str(path[-1])] = True


    if goalReached(path, goal_amount):
            found = True
            goal = path
            break
```

Ξεκινώντας, ορίζουμε το μέτωπο αναζήτησης (search_front), το οποίο είναι μια μορφή λίστας από τη βιβλιοθήκη collections ονόματι deque, για ευκολότερο χειρισμό της λίστας και αφαίρεση ή προσθήκη στοιχείων στο δεξί ή στο αριστερό της μέρος. Έπειτα, προσθέτουμε στο αριστερό μέρος της λίστας την αρχική κατάσταση η οποία είναι η initial_state = [[0, 0]]. Όσο το μέγεθος του μετώπου αναζήτησης δεν είναι 0, προσθέτουμε στο μονοπάτι path (μικροσκόπιο) το πρώτο στοιχείο του μετώπου αναζήτησης και ταυτόχρονα το αφαιρούμε από το ίδιο (χρησιμοποιώντας τη συνάρτηση popleft()). Ταυτόχρονα ορίζουμε ότι έχουμε επισκεφθεί την τρέχουσα κατάσταση μέσα στο dictionary visited ώστε να μην την ξαναεπισκεφθούμε και καταλήγουμε σε ατέρμονα βρόγχο. Στη συνέχεια, γίνεται έλεγχος αν η τρέχουσα κατάσταση η οποία είναι η τελευταία υπολίστα της λίστας path, είναι goal state. Σε περίπτωση που είναι, η μεταβλητή found γίνεται True, η αναζήτηση τερματίζει και τελικά εμφανίζεται το μονοπάτι το οποίο ακολουθήσαμε μέχρι να φτάσουμε στην τελική κατάσταση. Διαφορετικά σε περίπτωση που δεν έχουμε φτάσει ακόμα στο goal state, εκτελείται ο παρακάτω κώδικας:

```
next_moves = createChildren(jugs_capacities, path, visited)
        for i in next_moves:
            if choice:
                # implementing DFS
                search_front.appendleft(i)
            else:
                search_front.append(i)
```

Στο παραπάνω κομμάτι κώδικα γίνονται τα εξής βήματα: Αρχικά ορίζουμε ότι next_moves = createChildren(jugs_capacities, path, visited), δηλαδή η next_moves είναι πλέον μια λίστα από τα πιθανά μονοπάτια που μπορούμε να ακολουθήσουμε ξεκινώντας από την τρέχουσα κατάσταση (η λίστα possible_paths) που επιστρέφει η συνάρτηση createChildren(). Για κάθε στοιχείο που είναι μέσα σε αυτή τη λίστα (για κάθε πιθανό μονοπάτι δηλαδή), προσθέτουμε το μονοπάτι αυτό είτε στο αριστερό μέρος του μετώπου αναζήτησης searchFront είτε στο δεξί κομμάτι ανάλογα με την επιλογή που έχει κάνει ο χρήστης για τον αλγόριθμο αναζήτησης. Αφού προστεθούν όλα τα πιθανά μονοπάτια, επαναλαμβάνεται η παραπάνω διαδικασία (path = searchFront.popleft()……) εώς ότου φτάσουμε στην τελική επιθυμητή κατάσταση.

```
    if found:
        print("-----
THE GOAL HAS BEEN ACHIEVED, PRINTING THE PATH FOLLOWED TO THE GOAL STATE\n")
        for i in range(0, len(goal)):
            print(i, ". ", goal[i])
    else:
        print("-----THE PROBLEM CANNOT BE SOLVED, SORRY\n")
```

Φτάνοντας στην τελική κατάσταση, τυπώνεται το μονοπάτι που ακολουθήσαμε μέχρι να φτάσουμε στην τελική κατάσταση. Σε περίπτωση που δεν μπορεί να βρεθεί λύση στο πρόβλημα τυπώνεται ‘’-----THE PROBLEM CANNOT BE SOLVED, SORRY’’

Όλες οι παραπάνω συναρτήσεις και ο αλγόριθμος ξεκινούν να εκτελούνται αφού η συνάρτηση main() του κώδικα καλέσει αρχικά τη συνάρτηση search() δίνοντας της τα κατάλληλα και επιθυμητά ορίσματα, τα οποία έχουν οριστεί με βάση τις ανάγκες του προβλήματος, ωστόσο μπορούν να αλλαχθούν ώστε να επιλύεται το πρόβλημα και με διαφορετικές τιμές.

```python
def main():
    """
    Main function of the code
    """

    initial_state = [[0, 0]]        # initially both jugs are empty
    jugs_capacities = [70, 50]      # list to store the max capacities of the jugs
    goal_amount = 40                # the wanted amount of water in a jug
    visited = {}                    # dictionary to store the visited states
    choice = choose()               # choice calls def choose, if the user chooses DFS choice = True, ot
herwise choice = False
    search(initial_state, jugs_capacities, goal_amount, visited, choice)
```

Παρακάτω παρατίθονται παραδείγματα εκτέλεσης του κώδικα:

BFS

```
PS C:\Users\golem\OneDrive - unipi.gr\Python> & C:/Users/golem/AppData/Local/Programs/Python/Python39/python.exe
-----CHOOSE THE ALGORITHM YOU WANT TO USE. ENTER 'B' FOR BFS, 'D' FOR DFS:
b
-----IMPLEMENTING BFS

-----THE GOAL HAS BEEN ACHIEVED, PRINTING THE PATH FOLLOWED TO THE GOAL STATE

0 .  [0, 0]
1 .  [70, 0]
2 .  [20, 50]
3 .  [20, 0]
4 .  [0, 20]
5 .  [70, 20]
6 .  [40, 50]
PS C:\Users\golem\OneDrive - unipi.gr\Python>
```

DFS

```
PS C:\Users\golem\OneDrive - unipi.gr\Python> & C:/Users/golem/AppData/Local/Programs/Python/Python39/python.exe
-----CHOOSE THE ALGORITHM YOU WANT TO USE. ENTER 'B' FOR BFS, 'D' FOR DFS:
d
-----IMPLEMENTING DFS

-----THE GOAL HAS BEEN ACHIEVED, PRINTING THE PATH FOLLOWED TO THE GOAL STATE

0 .  [0, 0]
1 .  [0, 50]
2 .  [50, 0]
3 .  [50, 50]
4 .  [70, 30]
5 .  [70, 0]
6 .  [20, 50]
7 .  [20, 0]
8 .  [0, 20]
9 .  [70, 20]
10 .  [40, 50]
PS C:\Users\golem\OneDrive - unipi.gr\Python> []
```

Εκτέλεση αλλάζοντας ενδεικτικά τις ποσότητες:

```
initial_state = [[0, 0]]        # initially both jugs are empty
jugs_capacities = [30, 20]      # list to store the max capacities of the jugs
goal_amount = 70                # the wanted amount of water in a jug
visited = {}                    # dictionary to store the visited states
choice = choose()               # choice calls def choose, if the user chooses DFS choice = True, otherwise choice = False
search(initial_state, jugs_capacities, goal_amount, visited, choice)
```

```
PS C:\Users\golem\OneDrive - unipi.gr\Python> & C:/Users/golem/AppData/Local/Programs/Python/Python39/python.exe
-----CHOOSE THE ALGORITHM YOU WANT TO USE. ENTER 'B' FOR BFS, 'D' FOR DFS:
d
-----IMPLEMENTING DFS

-----THE GOAL HAS BEEN ACHIEVED, PRINTING THE PATH FOLLOWED TO THE GOAL STATE

0 .  [0, 0]
1 .  [0, 50]
2 .  [50, 0]
3 .  [50, 50]
4 .  [70, 30]
PS C:\Users\golem\OneDrive - unipi.gr\Python>
```