Τεχνητή Νοημοσύνη

2η άσκηση

Δεμερτζόγλου Ευστράτιος | TH20580

Table of Contents

Περιγραφή του Προβλήματος

Το πρόβλημα αφορά τη στοίβαξη τριών κουτιών a, b, c πάνω σε ένα τραπέζι t. Δίνεται μία **αρχική κατάσταση** και επιθυμούμε να φτάσουμε σε μία **τελική κατάσταση** χρησιμοποιώντας έναν και μόνο τελεστή:
move(X,Y)
που σημαίνει "μετακίνησε το αντικείμενο X πάνω στο Y".

Ο τελεστής έχει τις εξής **προϋποθέσεις**:

- Το X να μην έχει άλλο αντικείμενο από πάνω του.
- Το Y, αν δεν είναι το τραπέζι t, να μην έχει επίσης άλλο αντικείμενο από πάνω του.
- Η νέα κατάσταση να μην έχει ήδη επισκεφθεί ξανά στο παρελθόν (για αποφυγή βρόχων).

Προσέγγιση Επίλυσης

Το πρόβλημα επιλύεται με αναζήτηση στο **χώρο καταστάσεων**, χρησιμοποιώντας δύο κλασικούς αλγορίθμους τεχνητής νοημοσύνης:

- **Αναζήτηση Πρώτα κατά Πλάτος (BFS)**: Εξετάζει όλες τις πιθανές κινήσεις επιπέδου πριν προχωρήσει βαθύτερα.
- **Αναζήτηση Πρώτα κατά Βάθος (DFS)**: Εξερευνά πρώτα τις πιο πρόσφατες διαδρομές και επιστρέφει πίσω όταν χρειάζεται.

Επεξήση Κώδικα

Κάθε κατάσταση αναπαρίσταται ως ένα λεξικό που δείχνει που βρίσκεται κάθε block

Π.χ.:

```
initial_state = {'a': 'c', 'b': 't', 'c': 't'}
```

Το a βρίσκεται πάνω στο c, το b βρίσκεται πάνω στο t και το c βρίσκεται πάνω στο t

Βασικές Συναρτήσεις:

- has_on_top(state, x): Ελέγχει αν υπάρχει άλλο μπλοκ πάνω στο x.
- successors(state): Επιστρέφει όλες τις έγκυρες μετακινήσεις από την παρούσα κατάσταση.
- is_goal(state): Ελέγχει αν η κατάσταση είναι η τελική.
- bfs(start): Υλοποίηση του BFS για εύρεση της συντομότερης λύσης.
- dfs(start): Υλοποίηση DFS για εύρεση κάποιας λύσης.

Παράδειγμα Εκτέλεσης

Αυτό το παράδειγμα είναι drop down menu γιατί έχει πολύ μεγάλη έκταση

BFS run

DFS run

Κώδικας σε Python που εκτελέστηκε

```
from collections import deque
import copy

# Ελέγχει αν ένα αντικείμενο έχει κάτι πάνω του
def has_on_top(state, x):
    return any(v == x for v in state.values())

# Παράγει όλες τις επόμενες έγκυρες καταστάσεις από την τρέχουσα
def successors(state):
    result = []
    blocks = ['a', 'b', 'c']
```

```
    support = ['a', 'b', 'c', 't']  # Πάνω σε ποιο αντικείμενο μπορεί να μπει ένα μπλοκ

    for x in blocks:
        if has_on_top(state, x):
            continue  # Αν το X έχει κάτι πάνω του, δεν μπορούμε να το μετακινήσουμε

        for y in support:
            if x == y:
                continue  # Δεν επιτρέπεται να βάλουμε ένα μπλοκ πάνω στον εαυτό του
            if y != 't' and has_on_top(state, y):
                continue  # Αν το Y είναι άλλο μπλοκ και έχει κάτι πάνω του, δεν επιτρέπεται

            new_state = copy.deepcopy(state)
            new_state[x] = y  # Μετακινούμε το X πάνω στο Y

            if new_state != state:
                result.append((f"move({x},{y})", new_state))  # Προσθέτουμε νέο βήμα και κατάσταση

    return result


# Ελέγχει αν η κατάσταση είναι η τελική
def is_goal(state):
    return state == {'a': 'b', 'b': 'c', 'c': 't'}

# Εκτυπώνει την κατάσταση
def print_state(state):
    print("Κατάσταση:")
    for block in sorted(state.keys()):
        print(f"  {block} πάνω σε {state[block]}")
    print()


# BFS αναζήτηση
def bfs(start):
    visited = set()
    queue = deque([(start, [])])  # (τρέχουσα κατάσταση, λίστα κινήσεων)

    while queue:
        state, path = queue.popleft()
```

```
        state_tuple = tuple(sorted(state.items()))
        if state_tuple in visited:
            continue
        visited.add(state_tuple)

        if is_goal(state):
            print("\n--- Τελική κατάσταση επιτεύχθηκε (BFS) ---\n")
            print_state(state)
            return path # Επιστροφή των βημάτων που οδηγήσαν στην τελική
κατάσταση

        print("Εξετάζουμε νέα κατάσταση:")
        print_state(state)

        for action, new_state in successors(state):
            print(f"Εκτελώ: {action}")
            print_state(new_state)
            queue.append((new_state, path + [action]))

    return None # Δεν βρέθηκε λύση


# DFS αναζήτηση
def dfs(start):
    visited = set()
    stack = [(start, [])]  # (τρέχουσα κατάσταση, λίστα κινήσεων)

    while stack:
        state, path = stack.pop()
        state_tuple = tuple(sorted(state.items()))
        if state_tuple in visited:
            continue
        visited.add(state_tuple)

        if is_goal(state):
            print("\n--- Τελική κατάσταση επιτεύχθηκε (DFS) ---\n")
            print_state(state)
            return path # Επιστροφή των βημάτων που οδηγήσαν στην τελική
κατάσταση

        print("Εξετάζουμε νέα κατάσταση:")
        print_state(state)

        for action, new_state in successors(state):
            print(f"Εκτελώ: {action}")
```

```
            print_state(new_state)
            stack.append((new_state, path + [action]))

    return None # Δεν βρέθηκε λύση


# Αρχική κατάσταση
initial_state = {'a': 'c', 'b': 't', 'c': 't'}

# Εκτέλεση BFS και DFS
print("BFS Solution:")
print(bfs(initial_state))

print("\nDFS Solution:")
print(dfs(initial_state))
```

Συμπέρασμα

Και ο BFS και ο DFS αλγόριθμος κατέληξαν στην καλύτερη λύση η οποία είναι

['move(a,t)', 'move(b,c)', 'move(a,b)']