Τεχνητή Νοημοσύνη

Ρόμπερτ Πολοβίνα – 23390338

Εξάμηνο 7ο

Απαλλακτική Εργασία Εξαμήνου – Ρομποτική Σκούπα

Έγγραφο Τεκμηρίωσης

2025-2026

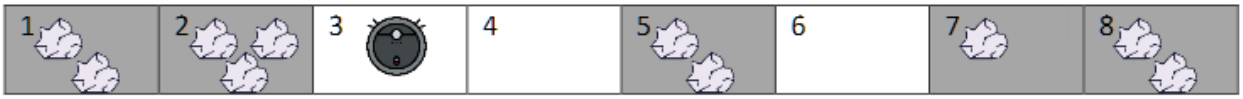
Περιεχόμενα

Περιγραφή του κόσμου και των καταστάσεων

Το πρόβλημα που εξετάζεται στην εργασία αφορά τον καθαρισμό ενός «διαδρόμου» από μια ρομποτική σκούπα. Ο διάδρομος αποτελείται από οκτώ πλακάκια πάνω στα οποία ενδέχεται να υπάρχουν ένα ή περισσότερα σκουπίδια.

Οι λειτουργίες που μπορεί να εκτελέσει η σκούπα είναι μετακίνηση προς τα δεξία ή τα αριστερά, να συλλέξει ένα έως τρία σκουπίδια και να τα αδειάσει όταν επιστρέφει στην βάση της.

Σκοπός της εργασίας είναι με δεδομένη μία **αρχική κατάσταση**, να βρεθεί μία ακολουθία ενεργειών (διαδρομή) που οδηγεί στην **τελική κατάσταση/στόχο,** δηλαδή όλα τα πλακάκια να είναι καθαρά και η σκούπα να βρίσκεται στην βάση της χωρίς φορτίο.



Η παραπάνω εικόνα αποτελεί την **αρχική κατάσταση** του προβλήματος και αναπαρίσταται ως εξής:

[3, 2, 3, 0, 0, 2, 0, 1, 8, 3, 0]

όπου:

[θέση σκούπας, σκουπίδια Ν πλακιδίου (1-8), θέση βάσης, φορτίο σκούπας]

και η **τελική κατάσταση** αναπαρίσταται ως:

[3, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0]

Ως **έγκυρες αρχικές καταστάσεις** δεχόμαστε τις καταστάσεις οι οποίες περιέχουν συνολικά 10 σκουπίδια, έχουν 8 πλακάκια, το πλακάκι στο οποίο βρίσκεται η σκούπα δεν έχει σκουπίδια και η σκούπα ξεκινάει από την βάση της χωρίς φορτίο. Για παράδειγμα άλλες έγκυρες καταστάσεις είναι:

- [4, 1, 0, 2, 0, 3, 3, 0, 1, 4, 0]
- [1, 0, 3, 0, 1, 1, 0, 2, 3, 1, 0]

- [7, 1, 3, 0, 1, 1, 2, 0, 2, 7, 0]
- [5, 1, 1, 2, 1, 0, 2, 0, 3, 5, 0]
- [8, 1, 2, 1, 0, 3, 3, 0, 0, 8, 0]

Αντιθέτως, οι **μη έγκυρες αρχικές καταστάσεις** δεν τηρούν αυτά τα κριτήρια όπως:

[1, 0, 1, 2, 1, 2, 3, 0, 1, 3, 0]: η σκούπα δεν ξεκινά από την βάση της

[2, 0, 1, 3, 2, 3, 0, 1, 0, 2, 0]: υπάρχουν σκουπίδια στην βάση της σκούπας

[3, 3, 1, 0, 2, 3, 0, 1, 0, 3, 2]: η σκούπα ξεκινάει με φορτίο

[5, 3, 1, 0, 2, 0, 5, 0]: ελλιπής αριθμός πλακιδίων

[7, 3, 1, 0, 0, 2, 0, 0, 0, 7, 0]: ελλιπής αριθμός σκουπιδιών

Τελεστές Μετάβασης, Απόγονοι και Στόχος

Οι **τελεστές μετάβασης** είναι οι τρεις λειτουργίες της σκούπας, δηλαδή μετακίνηση δεξιά & αριστερά και άδειασμα των σκουπιδιών. Η συλλογή σκουπιδιών συμπεριλαμβάνεται στην μετακίνηση. Μαζί με την συνάρτηση *find_children(),* καθορίζουν την μετάβαση από μία κατάσταση στην επόμενη και η συνάρτηση *is_goal_state()* είναι αυτή που θα αποφανθεί αν η τρέχουσα κατάσταση είναι και η τελική κατάσταση που ψάχνουμε. Ο κώδικας των συναρτήσεων έχει βασιστεί στον κώδικα που έχει δοθεί στα έγγραφα του μαθήματος. Συγκεκριμένα:

- **move_left(state):** μετακινεί την σκούπα ένα πλακάκι αριστερά
- **move_right(state):** μετακινεί την σκούπα ένα πλακάκι δεξιά
- **full(state):** η σκούπα πηγαίνει απευθείας στην βάση της και αδειάζει το φορτίο της μόνο όταν είναι γεμάτη ή δεν υπάρχουν άλλα σκουπίδια
- **is_goal_state(state):** βλέπει αν η τρέχουσα κατάσταση είναι και η τελική ζητούμενη
- **find_children(state):** δημιουργεί όλες τις δυνατές επόμενες καταστάσεις (απόγονοι)

- Οι τελεστές ***move_right()*** και ***move_left()***:
    1. Ελέγχουν τα όρια του διαδρόμου και το φορτίο της σκούπας
    2. Αν η σκούπα βρεθεί σε πλακάκι με σκουπίδια, συλλέγει όσα μπορεί για να γεμίσει
    3. Ενημερώνονται το πεδίο του φορτίου της σκούπας και το πλήθος των σκουπιδιών που βρίσκονται στο πλακάκι και επιστρέφεται η νέα κατάσταση.

```
#Μετακίνηση αριστερά
def move_left(state):
    #Η σκούπα θα μετακινηθεί μόνο αν δεν βρίσκεται στο αριστερό άκρο
    #Αν είναι γεμάτη μπλοκάρεται η μετακίνηση της για να τηλεμεταφερθεί στην βάση
    if state[-1]< 3 and state[0]> 1:
        state[0]= state[0]-1
        #Έλεγχος ύπαρξης σκουπιδιών στο νέο πλακάκι
        if state[state[0]]> 0:
            #Αν τα σκουπίδια ξεπερνούν την διαθέσιμη χωρητικότητα της σκούπας
            if state[state[0]]> 3-state[-1]:
                #μαζεύει όσα χωράνε για να γεμίσει
                state[state[0]]= state[state[0]] - (3-state[-1])
                state[-1]= 3
            else:
                #αλλιώς τα μαζεύει όλα
                state[-1]= state[-1] + state[state[0]]
                state[state[0]]= 0
    #Επιστρέφεται η νέα κατάσταση
    return state
#Μετακίνηση δεξιά
def move_right(state):
    #Θα μετακινηθεί μόνο αν δεν βρίσκεται στο δεξί άκρο
    #και άν δεν είναι γεμάτη
    if state[-1]< 3 and state[0]< 8:
        state[0]= state[0]+1
        #Υπόλοιπα βήματα ίδια με παραπάνω
        if state[state[0]]> 0:
            if state[state[0]]> 3- state[-1]:
                state[state[0]]-= (3- state[-1])
                state[-1]= 3
            else:
                state[-1]= state[-1] + state[state[0]]
                state[state[0]]= 0
    return state
```

- Ο τελεστής ***full()***:
    1. Στέλνει την σκούπα απευθείας στην βάση της όταν γεμίζει και την αδειάζει
    2. Σε άλλες περιπτώσεις δεν αδειάζει

```
#Άδειασμα σκουπιδιών μόνο όταν γεμίζει, με τηλεμεταφορά στην βάση
def full(state):
    load= state[-1]  #φορτίο σκούπας
    tiles_garbage= sum(state[1:9])  #συνολικά σκουπίδια στα πλακάκια (1-8)

    if load== 3 or tiles_garbage== 0:
        #Τηλεμεταφορά στη βάση
        state[0]= state[-2]   #θέση σκούπας= θέση βάσης
        state[-1]= 0      #άδειασμα φορτίου
        return state

    #Σε κάθε άλλη περίπτωση δεν αδειάζει
    else: return None
```

- Μέσω της συνάρτησης ***is_goal_state()***:

  βλέπουμε αν η τρέχουσα κατάσταση είναι και η τελική ζητούμενη για να σταματήσουμε την αναζήτηση.

  Για να ισχύει πρέπει όλα τα πλακάκια της τρέχουσας κατάστασης να μην περιέχουν σκουπίδια και η σκούπα να βρίσκεται στην βάση της χωρίς φορτίο. Με αυτόν τον τρόπο μπορούμε να εισάγουμε όποια αρχική κατάσταση θέλουμε χωρίς να χρειάζεται κάθε φορά να εισάγουμε ρητά και την τελική κατάσταση.

```
#Ελεγχος τελικής κατάστασης
def is_goal_state(state):
    #Για να επιλυθεί το πρόβλημα θα πρέπει:
    #1. Όλα τα πλακάκια 1-8 να έχουν 0 σκουπίδια
    clean= all(s== 0 for s in state[1:9])
    #2. Η σκούπα να είναι άδεια και στην βάση της
    back_to_base= (state[0]== state[-2] and state[-1]== 0)

    return clean and back_to_base
```

- Η συνάρτηση εύρεσης απογόνων ***find_children()***:

  δημιουργεί όλες τις δυνατές επόμενες καταστάσεις που μπορούν να προκύψουν από την τρέχουσα, εφαρμόζοντας τους τελεστές μετάβασης.

  Για κάθε τελεστή δημιουργείται μία νέα πιθανή κατάσταση και οι έγκυρες καταστάσεις (όχι *None*) προστίθενται στην λίστα *children*. Το αποτέλεσμα είναι η δημιουργία μιας λίστας όλων των απογόνων που μπορούν να εξεταστούν στην επόμενη επανάληψη της αναζήτησης.

  Στην αρχή της, η συνάρτηση ελέγχει εαν η τρέχουσα κατάσταση είναι και η τελική. Αν είναι, δεν δημιουργεί απογόνους.

```
#Εύρεση απογόνων της τρέχουσας κατάστασης
def find_children(state):
    #Αν η κατάσταση είναι η τελική, δεν δημιουργούνται απόγονοι
    if is_goal_state(state):
        return []

    children=[]

    #Εφαρμογή move_left()
    left_state= copy.deepcopy(state)
    left_child= move_left(left_state)

    if left_child!= None:
        children.append(left_child)
```

```
    #Εφαρμογή move_right()
    right_state= copy.deepcopy(state)
    right_child= move_right(right_state)

    if right_child!= None:
        children.append(right_child)

    #Εφαρμογή full()
    empty_state= copy.deepcopy(state)
    empty_child= full(empty_state)

    if empty_child!= None:
        children.append(empty_child)

    return children
```

Επίλυση με αλγορίθμους αναζήτησης DFS & BFS

Και οι δύο αλγόριθμοι υλοποιούνται σε δύο δομές, το **μέτωπο (front)** και την **ουρά (queue)**. Το μέτωπο είναι μια λίστα που περιέχει μόνο τις τρέχουσες καταστάσεις που περιμένουν να εξεταστούν, ενώ η ουρά είναι και αυτή μια λίστα που όμως κρατάει τα πλήρη μονοπάτια που οδηγούν σε κάθε κατάσταση του μετώπου. Ο αλγόριθμος DFS προσθέτει τους απογόνους στην αρχή του μετώπου (LIFO), ενώ ο BFS στο τέλος (FIFO).

Ακολουθεί ο κώδικας των δομών ο οποίος βασίστηκε στον κώδικα από τα έγγραφα του μαθήματος και επεκτάθηκε για να περιλαμβάνει και τον BFS. Στον παρακάτω κώδικα έχουν επίσης προστεθεί η **παρακολούθηση μετώπου (ζητ. 3)** και η **παρακολούθηση της ουράς των μονοπατιών (ζητ. 6)**. Τυπώνονται με αρίθμηση έτσι ώστε να είναι πιο εύκολη η αντιστοίχιση μεταξύ τους. Για την ολοκληρωμένη **παράλληλη παρακολούθηση**, έχει προστεθεί και η εκτύπωση της κατάστασης που εξετάζεται από την συνάρτηση *find_solution()*.

Μέτωπο:

```
def make_front(state):
    return [state]

def expand_front(front, method):
    if method== 'DFS':
        if front:
            #Παρακολούθηση μετώπου
            print("\nΜέτωπο:")
            for i, state in enumerate(front):
                print(f"  {i+1}: {state}")
            ##
            node= front.pop(0)
            for child in find_children(node):
```

```
                front.insert(0,child)

    elif method== 'BFS':
        if front:
            #Παρακολούθηση μετώπου
            print("\nΜέτωπο:")
            for i, state in enumerate(front):
                print(f"  {i+1}: {state}")
            ##
            node= front.pop(0)
            for child in find_children(node):
                front.append(child)

    return front
```

Ουρά:

```
def make_queue(state):
    return [[state]]

def extend_queue(queue, method):

    if method== 'DFS':
        #Παρακολούθηση ουράς
        print("\nΟυρά μονοπατιών:")
        for i, path in enumerate(queue):
            print(f"  {i+1}: {path}\n")
        print("=======================================")
        ##
        node=queue.pop(0)
        queue_copy= copy.deepcopy(queue)
        children= find_children(node[-1])
        for child in children:
            path= copy.deepcopy(node)
            path.append(child)
            queue_copy.insert(0,path)

    elif method== 'BFS':
        #Παρακολούθηση ουράς
        print("\nΟυρά μονοπατιών:")
        for i, path in enumerate(queue):
                print(f"  {i+1}: {path}\n")
        print("=======================================")
        ##
        node= queue.pop(0)
        queue_copy= copy.deepcopy(queue)
        children= find_children(node[-1])
        for child in children:
            path= copy.deepcopy(node)
            path.append(child)
            queue_copy.append(path)

    return queue_copy
```

find_solution():

```
def find_solution(front, queue, closed, method):

    if not front:
        print('_NO_SOLUTION_FOUND_')
        print(f"Μέθοδος: {method}\n")

    #Αν η κατάσταση έχει ξαναεξεταστεί, παραλείπεται
    elif front[0] in closed:
        new_front= copy.deepcopy(front)
        new_front.pop(0)
        new_queue= copy.deepcopy(queue)
        new_queue.pop(0)
        find_solution(new_front, new_queue, closed, method)

    #Μορφοποίηση εξόδου για να τυπώνουμε το μονοπάτι της λύσης
    #και άλλες πληροφορίες με ευανάγνωστο τρόπο
    elif is_goal_state(front[0]):
        print("\n_GOAL_FOUND_")
        print(front[0])
        print(f"\nΜονοπάτι λύσης ({len(queue[0])-1} βήματα):")
        for step, s in enumerate(queue[0]):
            print(f"Βήμα {step}: Θέση {s[0]}, Φορτίο {s[-1]}, Σκουπίδια
{sum(s[1:9])}")
        print(f"\nΣυνολικές καταστάσεις που εξετάστηκαν: {len(closed)}")
        print(f"Μέθοδος: {method}\n")

    else:
        closed.append(front[0])
        print("\nΚατάσταση που εξετάζεται:") # <-
        print(front[0])
        front_copy= copy.deepcopy(front)
        front_children=expand_front(front_copy, method)
        queue_copy= copy.deepcopy(queue)
        queue_children= extend_queue(queue_copy, method)
        closed_copy= copy.deepcopy(closed)
        find_solution(front_children, queue_children, closed_copy, method)
```

Αποτελέσματα:

DFS:

```
_GOAL_FOUND_
[3, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0]

Μονοπάτι λύσης (23 βήματα):
Βήμα 0: Θέση 3, Φορτίο 0, Σκουπίδια 10
Βήμα 1: Θέση 4, Φορτίο 0, Σκουπίδια 10
Βήμα 2: Θέση 5, Φορτίο 2, Σκουπίδια 8
Βήμα 3: Θέση 6, Φορτίο 2, Σκουπίδια 8
Βήμα 4: Θέση 7, Φορτίο 3, Σκουπίδια 7
Βήμα 5: Θέση 3, Φορτίο 0, Σκουπίδια 7
Βήμα 6: Θέση 4, Φορτίο 0, Σκουπίδια 7
Βήμα 7: Θέση 5, Φορτίο 0, Σκουπίδια 7
Βήμα 8: Θέση 6, Φορτίο 0, Σκουπίδια 7
Βήμα 9: Θέση 7, Φορτίο 0, Σκουπίδια 7
Βήμα 10: Θέση 8, Φορτίο 2, Σκουπίδια 5
Βήμα 11: Θέση 7, Φορτίο 2, Σκουπίδια 5
Βήμα 12: Θέση 6, Φορτίο 2, Σκουπίδια 5
Βήμα 13: Θέση 5, Φορτίο 2, Σκουπίδια 5
Βήμα 14: Θέση 4, Φορτίο 2, Σκουπίδια 5
Βήμα 15: Θέση 3, Φορτίο 2, Σκουπίδια 5
Βήμα 16: Θέση 2, Φορτίο 3, Σκουπίδια 4
Βήμα 17: Θέση 3, Φορτίο 0, Σκουπίδια 4
Βήμα 18: Θέση 2, Φορτίο 2, Σκουπίδια 2
Βήμα 19: Θέση 1, Φορτίο 3, Σκουπίδια 1
Βήμα 20: Θέση 3, Φορτίο 0, Σκουπίδια 1
Βήμα 21: Θέση 2, Φορτίο 0, Σκουπίδια 1
Βήμα 22: Θέση 1, Φορτίο 1, Σκουπίδια 0
Βήμα 23: Θέση 3, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 39
Μέθοδος: DFS
```

Από την **εκτέλεση με DFS** βλέπουμε ότι:

- Ο αλγόριθμος εξέτασε 39 καταστάσεις
- Βρήκε τον στόχο σε 23 βήματα (στο βήμα 0 η σκούπα είναι στην αρχική κατάσταση)

Συνεπώς η μέθοδος DFS φτάνει γρήγορα σε λύση αλλά όχι (απαραίτητα) στην βέλτιστη, όπως θα δείξει η μέθοδος BFS παρακάτω.

BFS:

```
_GOAL_FOUND_
[3, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0]

Μονοπάτι λύσης (18 βήματα):
Βήμα 0: Θέση 3, Φορτίο 0, Σκουπίδια 10
Βήμα 1: Θέση 2, Φορτίο 3, Σκουπίδια 7
Βήμα 2: Θέση 3, Φορτίο 0, Σκουπίδια 7
Βήμα 3: Θέση 4, Φορτίο 0, Σκουπίδια 7
Βήμα 4: Θέση 5, Φορτίο 2, Σκουπίδια 5
Βήμα 5: Θέση 4, Φορτίο 2, Σκουπίδια 5
Βήμα 6: Θέση 3, Φορτίο 2, Σκουπίδια 5
Βήμα 7: Θέση 2, Φορτίο 2, Σκουπίδια 5
Βήμα 8: Θέση 1, Φορτίο 3, Σκουπίδια 4
Βήμα 9: Θέση 3, Φορτίο 0, Σκουπίδια 4
Βήμα 10: Θέση 4, Φορτίο 0, Σκουπίδια 4
Βήμα 11: Θέση 5, Φορτίο 0, Σκουπίδια 4
Βήμα 12: Θέση 6, Φορτίο 0, Σκουπίδια 4
Βήμα 13: Θέση 7, Φορτίο 1, Σκουπίδια 3
Βήμα 14: Θέση 8, Φορτίο 3, Σκουπίδια 1
Βήμα 15: Θέση 3, Φορτίο 0, Σκουπίδια 1
Βήμα 16: Θέση 2, Φορτίο 0, Σκουπίδια 1
Βήμα 17: Θέση 1, Φορτίο 1, Σκουπίδια 0
Βήμα 18: Θέση 3, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 109
Μέθοδος: BFS
```

Από την **εκτέλεση με BFS** βλέπουμε ότι:

- ο BFS εξέτασε πολλές παραπάνω καταστάσεις σε σχέση με τον DFS (109 έναντι 39)
- Βρήκε την βέλτιστη διαδρομή κατά 5 λιγότερα βήματα (18 έναντι 23).

Δέντρα Αναζήτησης DFS & BFS

Εδώ παρουσιάζεται σχηματικά, μετά την επεξήγηση, ο τρόπος λειτουργίας των δύο αλγορίθμων. Η δημιουργία δέντρου 4 επιπέδων με την αρχική κατάσταση της εκφώνησης οδηγεί σε πολλές παραπάνω καταστάσεις (>30) απ' όσες θα μπορούσαμε εύκολα να υπολογίσουμε και να απεικονίσουμε.

Συνεπώς θα προχωρήσουμε στην επίδειξη των αλγορίθμων αναζήτησης για την παρακάτω απλή αρχική κατάσταση, που λύνεται σε 4 επίπεδα. Για να μπορέσει αυτό το πολύ απλό παράδειγμα να μας φανεί χρήσιμο και να μην τελειώσει υπερβολικά πρόωρα, δεχόμαστε την παραδοχή πως η σκούπα με τον τελεστή ***full*** δεν γυρνάει απευθείας πίσω στην βάση της αλλά πρέπει να κάνει όλα τα βήματα κανονικά. Ως αρχική κατάσταση ορίζουμε:

[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0]

με κατάσταση στόχο:

[1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

Επίπεδο 0 (ρίζα):

Στην ρίζα του δέντρου βρίσκεται η αρχική κατάσταση την οποία ονομάζουμε ως S0.

Επίπεδο 1:

Από την αρχική κατάσταση (S0), υπάρχουν δύο πιθανές καταστάσεις:

S0 (δεν αλλάζει): *full()* -> [1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0]

S1: *move_right()* -> [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]

Η σκούπα δεν μπορεί να μετακινηθεί αριστερά (*move_left)* διότι βρίσκεται στην θέση 1 και αν το έκανε θα έβγαινε εκτός ορίων.

Επίπεδο 2:

Η σκούπα μετακινήθηκε δεξιά και σύλλεξε το σκουπίδι, δηλαδή βρισκόμαστε στην κατάσταση **S1.** Από την S1 υπάρχουν τρεις επιλογές:

S1 (δεν αλλάζει): *full()* -> [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]

S2: *move_left()* -> [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]

S3: *move_right()* -> [3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]

Πηγαίνοντας δεξιά η σκούπα θα βρεθεί σε περιττή κατάσταση (**S3**) η οποία δεν οδηγεί στον στόχο. Ο DFS όμως δεν μπορεί να το γνωρίζει αυτό. Εδώ είναι πολύ πιθανό να αρχίσει να ψάχνει και να επεκτείνει καταστάσεις από την S3 και μετά. Βλέπουμε λοιπόν ένα από τα μειονεκτήματα του DFS που είναι η σπατάλη για τον έλεγχο περιττών καταστάσεων μέχρι να επιστρέψει πάλι στο σωστό μονοπάτι. Συνεχίζουμε στο παράδειγμα μας με την παραδοχή πως ο αλγόριθμος συνέχισε στο μονοπάτι που οδηγεί στην λύση. Συνεπώς η σκούπα θα μετακινηθεί αριστερά και θα βρεθεί στην κατάσταση **S2**.

Επίπεδο 3:

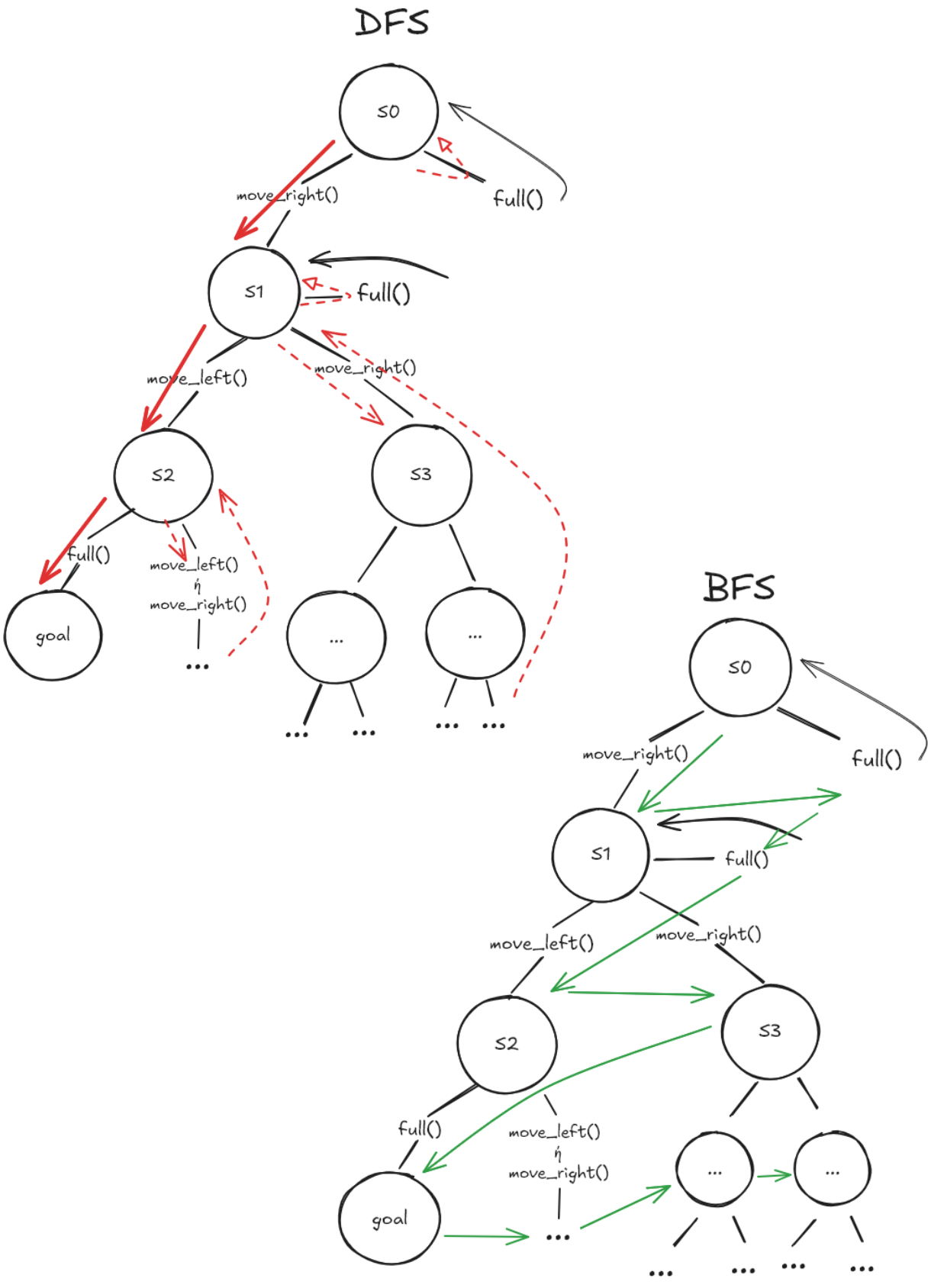
Η σκούπα βρίσκεται τώρα στην βάση της με φορτίο ένα σκουπίδι. Συνεπώς θα εκτελέσει την λειτουργία ***empty_robot*** και θα βρεθεί στην τελική κατάσταση λύνοντας το πρόβλημα, ή εκτελώντας ***move_left*** ή ***move_right*** θα παρεκτραπεί και θα απομακρυνθεί από την λύση:

goal: *full()* -> [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

περιττές καταστάσεις: *move_left() / move_right()* -> [...]

Πρακτικά, η μέθοδος DFS ακολουθεί μία διαδρομή από την ρίζα του δέντρου προς τα βάθη του, εξετάζοντας διαδοχικά τις επόμενες καταστάσεις που προκύπτουν από τις επιτρεπτές ενέργειες. Δεν εξετάζει δηλαδή ολόκληρο το επίπεδο πριν προχωρήσει. Από την άλλη, η BFS θα εξερευνήσει πρώτα όλες τις καταστάσεις ενός επιπέδου πριν περάσει στο επόμενο.

Το παραπάνω παράδειγμα απεικονίζεται σχηματικά με τα δέντρα που ακολουθούν. Η DFS θα ακολουθήσει το αριστερό μονοπάτι έως το τέλος του και θα βρει την λύση αμέσως ή θα παρεκτραπεί στην S3 και θα εξαντλήσει πρώτα εκείνο το μονοπάτι πριν επιστρέψει στο σωστό. Η BFS θα εξετάσει πρώτα όλους τους κόμβους του επιπέδου πριν προχωρήσει στο επόμενο και συνεπώς, στο πολύ απλό αυτό παράδειγμα, θα βρει πρώτη την λύση διότι δεν θα σπαταλήσει χρόνο εξερευνώντας άσκοπα το μονοπάτι της S3 .Σε πιο περίπλοκες συνθήκες όμως, όπως είδαμε στην προηγούμενη ενότητα, η DFS (συνήθως) θα εξετάζει πολλές λιγότερες καταστάσεις ενώ η BFS επειδή θα βλέπει την λύση όταν αυτή θα πρωτοεμφανίζεται στο δέντρο, δηλαδή στο συγκριτικά υψηλότερο επίπεδο, θα βρίσκει πάντα την βέλτιστη.


DFS
s0
move_right()
full()
s1
full()
move_left()
move_right()
s2
s3
full()
move_left()
ή
move_right()
goal
...
...
...
...
...
BFS
s0
move_right()
full()
s1
full()
move_left()
move_right()
s2
s3
full()
move_left()
ή
move_right()
goal
...
...
...
...
...
...

Ευριστική Μέθοδος Αναζήτησης

Οι μέθοδοι DFS και BFS που βλέπουμε μέχρι τώρα είναι «τυφλές» μέθοδοι αναζήτησης που απλώς ψάχνουν μέσα σε μία λίστα (δέντρο) μέχρι να βρουν την τελική κατάσταση. Δεν διαθέτουν κάποιον μηχανισμό για να κάνουν την αναζήτηση πιο γρήγορη και αποτελεσματική. Ο μηχανισμός αυτός ονομάζεται **Ευριστική Τιμή** και χρησιμοποιείται από τους ευριστικούς μηχανισμούς αναζήτησης για να κατευθύνονται πιο «έξυπνα» προς τον στόχο, μειώνοντας τον αριθμό των καταστάσεων που θα εξεταστούν και πλησιάζοντας την βέλτιστη λύση.

Στο πρόβλημα μας ως ευριστική τιμή τέθηκε το άθροισμα των υπολειπόμενων σκουπιδιών στον χώρο. Εφόσον η σκούπα πηγαίνει απευθείας στην βάση της όταν γεμίζει, δεν έχουμε λόγο να συμπεριλάβουμε στην ευριστική τιμή και την απόσταση από την βάση.

```
#Δημιουργία ευριστικής τιμής για μία κατάσταση
def heuristic(state):
    #Ευριστική τιμή= άθροισμα σκουπιδιών στον χώρο
    return sum(state[1:9])
```

Πρακτικά ενώ οι DFS και BFS βασίζονται στην σειρά των καταστάσεων μέσα σε δύο ξεχωριστές δομές (**front** και **queue**), η ευριστική αναζήτηση μας βασίζεται στην προτεραιότητα τους. Χρησιμοποιήθηκε η ευριστική μέθοδος **Best-First Search** η οποία χρειάζεται μία ενιαία δομή που θα πρέπει να κρατάει και να επιλέγει τις καταστάσεις με βάση την ευριστική τιμή που τις συνοδεύει. Σκοπός είναι η επόμενη κατάσταση που θα εξεταστεί να είναι αυτή με την μικρότερη ευριστική τιμή, δηλαδή με την μεγαλύτερη προτεραιότητα.

Συνεπώς, η ευριστική αναζήτηση υλοποιείται με τον παρακάτω κώδικα:

```
def best_first_search(initial_state):
    queue= []
    #Προσθήκη της αρχικής κατάστασης στην ουρά
    #μαζί με την ευριστική τιμή
    queue.append((heuristic(initial_state), [initial_state]))
    closed= []
    #Μετρητής καταστάσεων που εξετάστηκαν
    states_examined= 0

    while queue:
        #Βρίσκουμε την κατάσταση με την μικρότερη ευριστική τιμή
        #(μεγαλύτερη προτεραιότητα)
        min_index= 0
        min_h= queue[0][0]
        for i in range(1, len(queue)):
            if queue[i][0]< min_h:
                min_h= queue[i][0]
```

```
                min_index= i

        #Εξάγουμε την κατάσταση με την μεγαλύτερη προτεραιότητα
        #Αναθέτουμε την ευριστική τιμή της στην μεταβλητή h_value
        #και την ίδια την κατάσταση (μονοπάτι μέχρι την τρέχουσα) στην path
        h_value, path= queue.pop(min_index)
        current= path[-1]   #path[-1] -> η πιο πρόσφατη κατάσταση (τρέχουσα)

        #στην path_new θα αποθηκεύεται σταδιακά ολόκληρο το μονοπάτι της λύσης
        #για να τυπωθεί στο τέλος
        path_new= path

        #Αν η κατάσταση έχει εξεταστεί ξανά, παραλείπεται
        if current in closed:
            continue

        closed.append(current)

        print("\nΚατάσταση που εξετάζεται:")
        print(current)
        print(f"Ευριστική τιμή h= {h_value}")
        states_examined+= 1

        for child in find_children(current):
            if child not in closed:
                path_new= path+ [copy.deepcopy(child)]
                h_child= heuristic(child)
                #Προσθήκη παιδιού στην ουρά
                queue.append((h_child, path_new))

        #Ο έλεγχος του στόχου γίνεται εδώ και όχι πριν την δημιουργία απογόνων
        #γιατί για να τυπωθεί το μονοπάτι της λύσης χρειαζόμαστε ολοκληρωμένη την
path_new
        if is_goal_state(current):
            print("\n_GOAL_FOUND_")
            print(current)
            print(f"\nΜονοπάτι λύσης ({len(path_new)-1} βήματα):")
            for step, s in enumerate(path_new):
                print(f"Βήμα {step}: Θέση {s[0]}, Φορτίο {s[-1]}, Σκουπίδια
{sum(s[1:9])}")
            print(f"\nΣυνολικές καταστάσεις που εξετάστηκαν: {states_examined}")
            print("Μέθοδος: Ευριστική (Best-First Search)\n")

            return path_new


    print("_NO_SOLUTION_FOUND_")
    print("Μέθοδος: Ευριστική (Best-First Search)\n")
```

Η ***queue*** είναι μία λίστα και κάθε στοιχείο της αποτελείται από δύο τιμές, την ***h_value*** και την ***path***. Η ***h_value*** είναι η ευριστική τιμή της τρέχουσας κατάστασης και η μεταβλητή ***path*** είναι μία λίστα με τις καταστάσεις που αποτελούν το μονοπάτι το οποίο οδηγεί στην τρέχουσα κατάσταση, όπως και η ουρά στους DFS/BFS. Η ***closed*** είναι η λίστα στην οποία προσθέτουμε τις καταστάσεις που εξετάζουμε για να μην τυχόν επανεξεταστούν.

Για να εξετάσουμε την κατάσταση με την μεγαλύτερη προτεραιότητα, αναζητούμε κάθε φορά μέσα στην ***queue*** την κατάσταση με την μικρότερη ευριστική τιμή μέσω μίας επανάληψης ***for***. Όταν η κατάσταση αυτή βρεθεί διαχωρίζουμε τα περιεχόμενα της σε δύο μεταβλητές, ελέγχουμε αν η κατάσταση έχει επανεξεταστεί και αν όχι προχωράμε στην εύρεση απογόνων.

Στην εύρεση απογόνων (με χρήση της ***find_children***) βρίσκουμε επίσης και την ευριστική τιμή για κάθε παιδί, το οποίο έπειτα το προσθέτουμε στην ***queue***.

Τέλος ακολουθεί ο έλεγχος τελικής κατάστασης. Κανονικά πιο σωστό θα ήταν να γίνεται πριν την εύρεση απογόνων αλλά επειδή θέλουμε να τυπώσουμε το μονοπάτι που οδηγεί στην λύση χρειαζόμαστε την ***path_new*** να είναι ολοκληρωμένη, δηλαδή να έχει τελειώσει η διαδικασία εύρεσης απογόνων. Δεν μπορούμε να χρησιμοποιήσουμε την ***path*** διότι περιέχει μόνο το μονοπάτι για την τρέχουσα κατάσταση. Λόγω αυτής της ιδιαιτερότητας, έχουμε φροντίσει μέσα στην συνάρτηση ***find_children()*** να μην δημιουργούνται απόγονοι αν βρισκόμαστε στην τελική κατάσταση, όπως έχει ήδη δειχθεί στον κώδικα της (σελ. 5-6).

Αν η τρέχουσα κατάσταση είναι και η τελική, λήγουμε την αναζήτηση αφού τυπώσουμε το μονοπάτι της λύσης. Αλλιώς πάμε πάλι στην αρχή της επανάληψης και ξεκινάει η διαδικασία εκ νέου, αναζητώντας την επόμενη κατάσταση με την μεγαλύτερη προτεραιότητα.

```
_GOAL_FOUND_
[3, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0]

Μονοπάτι λύσης (21 βήματα):
Βήμα 0: Θέση 3, Φορτίο 0, Σκουπίδια 10
Βήμα 1: Θέση 2, Φορτίο 3, Σκουπίδια 7
Βήμα 2: Θέση 3, Φορτίο 0, Σκουπίδια 7
Βήμα 3: Θέση 2, Φορτίο 0, Σκουπίδια 7
Βήμα 4: Θέση 1, Φορτίο 2, Σκουπίδια 5
Βήμα 5: Θέση 2, Φορτίο 2, Σκουπίδια 5
Βήμα 6: Θέση 3, Φορτίο 2, Σκουπίδια 5
Βήμα 7: Θέση 4, Φορτίο 2, Σκουπίδια 5
Βήμα 8: Θέση 5, Φορτίο 3, Σκουπίδια 4
Βήμα 9: Θέση 3, Φορτίο 0, Σκουπίδια 4
Βήμα 10: Θέση 4, Φορτίο 0, Σκουπίδια 4
Βήμα 11: Θέση 5, Φορτίο 1, Σκουπίδια 3
Βήμα 12: Θέση 6, Φορτίο 1, Σκουπίδια 3
Βήμα 13: Θέση 7, Φορτίο 2, Σκουπίδια 2
Βήμα 14: Θέση 8, Φορτίο 3, Σκουπίδια 1
Βήμα 15: Θέση 3, Φορτίο 0, Σκουπίδια 1
Βήμα 16: Θέση 4, Φορτίο 0, Σκουπίδια 1
Βήμα 17: Θέση 5, Φορτίο 0, Σκουπίδια 1
Βήμα 18: Θέση 6, Φορτίο 0, Σκουπίδια 1
Βήμα 19: Θέση 7, Φορτίο 0, Σκουπίδια 1
Βήμα 20: Θέση 8, Φορτίο 1, Σκουπίδια 0
Βήμα 21: Θέση 3, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 27
Μέθοδος: Ευριστική (Best-First Search)
```

Η ευριστική αναζήτηση σε σχέση με την **DFS**:

- Βρήκε καλύτερη λύση με 2 λιγότερα βήματα (21 έναντι 23)
- Εξέτασε 12 λιγότερες καταστάσεις (27 έναντι 39)

Συγκριτικά με την **BFS**:

- Η ευριστική δεν βρίσκει την βέλτιστη λύση που βρήκε η BFS (21 βήματα έναντι 18)
- Εξετάζει πάρα πολλές λιγότερες καταστάσεις (27 έναντι 109)
- Εδώ να σημειώσουμε πως η Best-First Search μέθοδος δεν εγγυάται την εύρεση της βέλτιστης λύσης για κάθε αρχική κατάσταση, αλλά μία πιο στοχευμένη προσέγγιση όσον αφορά την διαδικασία της αναζήτησης.

Παράλληλη παρακολούθηση και επιλογή μεθόδων

Όπως είδαμε και παραπάνω στην παρουσίαση του κώδικα του μετώπου και της ουράς (σελ. 6-7), η παράλληλη παρακολούθηση έχει υλοποιηθεί αλλά αναφέρεται κι εδώ για λόγους πληρότητας.

```
#Παρακολούθηση μετώπου (expand_front())
            print("\nΜέτωπο:")
            for i, state in enumerate(front):
                print(f"  {i+1}: {state}")
            ##
        …
        …

#Παρακολούθηση ουράς (extend_queue())
        print("\nΟυρά μονοπατιών:")
        for i, path in enumerate(queue):
            print(f"  {i+1}: {path}\n")
        print("=====================================")
        ##
       …
       …

#Εκτύπωση τρέχουσας κατάστασης που εξετάζεται (find_solution())

      …
else:
        closed.append(front[0])
        print("\nΚατάσταση που εξετάζεται:")
        print(front[0])
        …
        …
```

Στο κύριο πρόγραμμα έχει επίσης προστεθεί και η δυνατότητα επιλογής μεταξύ των τριών μεθόδων αναζήτησης.

```
def main():

    #[θέση σκούπας, σκουπίδια Ν πλακιδίου (1-8), θέση βάσης, φορτίο σκούπας]
    initial_state = [3, 2, 3, 0, 0, 2, 0, 1, 2, 3, 0]
    #goal = [3, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0]
    #Ο καθορισμός της τελικής κατάστασης δεν είναι απαραίτητος

    while True:
        print("\nΕπιλέξτε μέθοδο αναζήτησης ή 0 για έξοδο.")
        print("1 - DFS\n2 - BFS\n3 - Ευριστική (Best-First)")
        search= input("Επιλογή: ")

        if search== '1':
            method= 'DFS'
            print('\n____BEGIN__SEARCHING____')
            find_solution(make_front(initial_state), make_queue(initial_state), [],
method)
        elif search== '2':
            method= 'BFS'
            print('\n____BEGIN__SEARCHING____')
            find_solution(make_front(initial_state), make_queue(initial_state), [],
method)
        elif search== '3':
            print('\n____BEGIN__SEARCHING____')
            best_first_search(initial_state)
        elif search== '0':
            break
        else:
            print("Μη έγκυρη επιλογή")


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

Συγκριτική μελέτη αποτελεσμάτων

Κατάσταση 1

Έστω η αρχική κατάσταση:

[4, 1, 0, 2, 0, 3, 3, 0, 1, 4, 0]

DFS:

```
_GOAL_FOUND_
[4, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0]

Μονοπάτι λύσης (19 βήματα):
Βήμα 0: Θέση 4, Φορτίο 0, Σκουπίδια 10
Βήμα 1: Θέση 5, Φορτίο 3, Σκουπίδια 7
Βήμα 2: Θέση 4, Φορτίο 0, Σκουπίδια 7
Βήμα 3: Θέση 5, Φορτίο 0, Σκουπίδια 7
Βήμα 4: Θέση 6, Φορτίο 3, Σκουπίδια 4
Βήμα 5: Θέση 4, Φορτίο 0, Σκουπίδια 4
Βήμα 6: Θέση 5, Φορτίο 0, Σκουπίδια 4
Βήμα 7: Θέση 6, Φορτίο 0, Σκουπίδια 4
Βήμα 8: Θέση 7, Φορτίο 0, Σκουπίδια 4
Βήμα 9: Θέση 8, Φορτίο 1, Σκουπίδια 3
Βήμα 10: Θέση 7, Φορτίο 1, Σκουπίδια 3
Βήμα 11: Θέση 6, Φορτίο 1, Σκουπίδια 3
Βήμα 12: Θέση 5, Φορτίο 1, Σκουπίδια 3
Βήμα 13: Θέση 4, Φορτίο 1, Σκουπίδια 3
Βήμα 14: Θέση 3, Φορτίο 3, Σκουπίδια 1
Βήμα 15: Θέση 4, Φορτίο 0, Σκουπίδια 1
Βήμα 16: Θέση 3, Φορτίο 0, Σκουπίδια 1
Βήμα 17: Θέση 2, Φορτίο 0, Σκουπίδια 1
Βήμα 18: Θέση 1, Φορτίο 1, Σκουπίδια 0
Βήμα 19: Θέση 4, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 23
Μέθοδος: DFS
```

- Μονοπάτι λύσης: 19 βήματα
- Καταστάσεις που εξετάστηκαν: 23

BFS:

```
_GOAL_FOUND_
[4, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0]

Μονοπάτι λύσης (14 βήματα):
Βήμα 0: Θέση 4, Φορτίο 0, Σκουπίδια 10
Βήμα 1: Θέση 3, Φορτίο 2, Σκουπίδια 8
Βήμα 2: Θέση 2, Φορτίο 2, Σκουπίδια 8
Βήμα 3: Θέση 1, Φορτίο 3, Σκουπίδια 7
Βήμα 4: Θέση 4, Φορτίο 0, Σκουπίδια 7
Βήμα 5: Θέση 5, Φορτίο 3, Σκουπίδια 4
Βήμα 6: Θέση 4, Φορτίο 0, Σκουπίδια 4
Βήμα 7: Θέση 5, Φορτίο 0, Σκουπίδια 4
Βήμα 8: Θέση 6, Φορτίο 3, Σκουπίδια 1
Βήμα 9: Θέση 4, Φορτίο 0, Σκουπίδια 1
Βήμα 10: Θέση 5, Φορτίο 0, Σκουπίδια 1
Βήμα 11: Θέση 6, Φορτίο 0, Σκουπίδια 1
Βήμα 12: Θέση 7, Φορτίο 0, Σκουπίδια 1
Βήμα 13: Θέση 8, Φορτίο 1, Σκουπίδια 0
Βήμα 14: Θέση 4, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 71
Μέθοδος: BFS
```

- Μονοπάτι λύσης: 14 βήματα
- Καταστάσεις που εξετάστηκαν: 71

Best-First Search:

```
_GOAL_FOUND_
[4, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0]

Μονοπάτι λύσης (14 βήματα):
Βήμα 0: Θέση 4, Φορτίο 0, Σκουπίδια 10
Βήμα 1: Θέση 5, Φορτίο 3, Σκουπίδια 7
Βήμα 2: Θέση 4, Φορτίο 0, Σκουπίδια 7
Βήμα 3: Θέση 3, Φορτίο 2, Σκουπίδια 5
Βήμα 4: Θέση 2, Φορτίο 2, Σκουπίδια 5
Βήμα 5: Θέση 1, Φορτίο 3, Σκουπίδια 4
Βήμα 6: Θέση 4, Φορτίο 0, Σκουπίδια 4
Βήμα 7: Θέση 5, Φορτίο 0, Σκουπίδια 4
Βήμα 8: Θέση 6, Φορτίο 3, Σκουπίδια 1
Βήμα 9: Θέση 4, Φορτίο 0, Σκουπίδια 1
Βήμα 10: Θέση 5, Φορτίο 0, Σκουπίδια 1
Βήμα 11: Θέση 6, Φορτίο 0, Σκουπίδια 1
Βήμα 12: Θέση 7, Φορτίο 0, Σκουπίδια 1
Βήμα 13: Θέση 8, Φορτίο 1, Σκουπίδια 0
Βήμα 14: Θέση 4, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 20
Μέθοδος: Ευριστική (Best-First Search)
```

- Μονοπάτι λύσης: 14 βήματα
- Καταστάσεις που εξετάστηκαν: 20

Από τα αποτελέσματα βλέπουμε πως η ευριστική αναζήτηση σε αυτό το παράδειγμα βρίσκει την βέλτιστη λύση που βρήκε και η BFS εξετάζοντας όμως λιγότερες καταστάσεις ακόμα και από την DFS. Αποδίδει δηλαδή καλύτερα από τις τυφλές μεθόδους και για τα δύο κριτήρια.

Κατάσταση 2

Έστω η κατάσταση:

[1, 0, 3, 0, 1, 1, 0, 2, 3, 1, 0]

DFS:

```
_GOAL_FOUND_
[1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

Μονοπάτι λύσης (25 βήματα):
Βήμα 0: Θέση 1, Φορτίο 0, Σκουπίδια 10
Βήμα 1: Θέση 2, Φορτίο 3, Σκουπίδια 7
Βήμα 2: Θέση 1, Φορτίο 0, Σκουπίδια 7
Βήμα 3: Θέση 2, Φορτίο 0, Σκουπίδια 7
Βήμα 4: Θέση 3, Φορτίο 0, Σκουπίδια 7
Βήμα 5: Θέση 4, Φορτίο 1, Σκουπίδια 6
Βήμα 6: Θέση 5, Φορτίο 2, Σκουπίδια 5
Βήμα 7: Θέση 6, Φορτίο 2, Σκουπίδια 5
Βήμα 8: Θέση 7, Φορτίο 3, Σκουπίδια 4
Βήμα 9: Θέση 1, Φορτίο 0, Σκουπίδια 4
Βήμα 10: Θέση 2, Φορτίο 0, Σκουπίδια 4
Βήμα 11: Θέση 3, Φορτίο 0, Σκουπίδια 4
Βήμα 12: Θέση 4, Φορτίο 0, Σκουπίδια 4
Βήμα 13: Θέση 5, Φορτίο 0, Σκουπίδια 4
Βήμα 14: Θέση 6, Φορτίο 0, Σκουπίδια 4
Βήμα 15: Θέση 7, Φορτίο 1, Σκουπίδια 3
Βήμα 16: Θέση 8, Φορτίο 3, Σκουπίδια 1
Βήμα 17: Θέση 1, Φορτίο 0, Σκουπίδια 1
Βήμα 18: Θέση 2, Φορτίο 0, Σκουπίδια 1
Βήμα 19: Θέση 3, Φορτίο 0, Σκουπίδια 1
Βήμα 20: Θέση 4, Φορτίο 0, Σκουπίδια 1
Βήμα 21: Θέση 5, Φορτίο 0, Σκουπίδια 1
Βήμα 22: Θέση 6, Φορτίο 0, Σκουπίδια 1
Βήμα 23: Θέση 7, Φορτίο 0, Σκουπίδια 1
Βήμα 24: Θέση 8, Φορτίο 1, Σκουπίδια 0
Βήμα 25: Θέση 1, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 25
Μέθοδος: DFS
```

- Μονοπάτι λύσης: 25 βήματα
- Καταστάσεις που εξετάστηκαν: 25

BFS:

```
_GOAL_FOUND_
[1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

Μονοπάτι λύσης (25 βήματα):
Βήμα 0: Θέση 1, Φορτίο 0, Σκουπίδια 10
Βήμα 1: Θέση 2, Φορτίο 3, Σκουπίδια 7
Βήμα 2: Θέση 1, Φορτίο 0, Σκουπίδια 7
Βήμα 3: Θέση 2, Φορτίο 0, Σκουπίδια 7
Βήμα 4: Θέση 3, Φορτίο 0, Σκουπίδια 7
Βήμα 5: Θέση 4, Φορτίο 1, Σκουπίδια 6
Βήμα 6: Θέση 5, Φορτίο 2, Σκουπίδια 5
Βήμα 7: Θέση 6, Φορτίο 2, Σκουπίδια 5
Βήμα 8: Θέση 7, Φορτίο 3, Σκουπίδια 4
Βήμα 9: Θέση 1, Φορτίο 0, Σκουπίδια 4
Βήμα 10: Θέση 2, Φορτίο 0, Σκουπίδια 4
Βήμα 11: Θέση 3, Φορτίο 0, Σκουπίδια 4
Βήμα 12: Θέση 4, Φορτίο 0, Σκουπίδια 4
Βήμα 13: Θέση 5, Φορτίο 0, Σκουπίδια 4
Βήμα 14: Θέση 6, Φορτίο 0, Σκουπίδια 4
Βήμα 15: Θέση 7, Φορτίο 1, Σκουπίδια 3
Βήμα 16: Θέση 8, Φορτίο 3, Σκουπίδια 1
Βήμα 17: Θέση 1, Φορτίο 0, Σκουπίδια 1
Βήμα 18: Θέση 2, Φορτίο 0, Σκουπίδια 1
Βήμα 19: Θέση 3, Φορτίο 0, Σκουπίδια 1
Βήμα 20: Θέση 4, Φορτίο 0, Σκουπίδια 1
Βήμα 21: Θέση 5, Φορτίο 0, Σκουπίδια 1
Βήμα 22: Θέση 6, Φορτίο 0, Σκουπίδια 1
Βήμα 23: Θέση 7, Φορτίο 0, Σκουπίδια 1
Βήμα 24: Θέση 8, Φορτίο 1, Σκουπίδια 0
Βήμα 25: Θέση 1, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 39
Μέθοδος: BFS
```

- Μονοπάτι λύσης: 25 βήματα
- Καταστάσεις που εξετάστηκαν: 39

Best-First Search:

```
_GOAL_FOUND_
[1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

Μονοπάτι λύσης (25 βήματα):
Βήμα 0: Θέση 1, Φορτίο 0, Σκουπίδια 10
Βήμα 1: Θέση 2, Φορτίο 3, Σκουπίδια 7
Βήμα 2: Θέση 1, Φορτίο 0, Σκουπίδια 7
Βήμα 3: Θέση 2, Φορτίο 0, Σκουπίδια 7
Βήμα 4: Θέση 3, Φορτίο 0, Σκουπίδια 7
Βήμα 5: Θέση 4, Φορτίο 1, Σκουπίδια 6
Βήμα 6: Θέση 5, Φορτίο 2, Σκουπίδια 5
Βήμα 7: Θέση 6, Φορτίο 2, Σκουπίδια 5
Βήμα 8: Θέση 7, Φορτίο 3, Σκουπίδια 4
Βήμα 9: Θέση 1, Φορτίο 0, Σκουπίδια 4
Βήμα 10: Θέση 2, Φορτίο 0, Σκουπίδια 4
Βήμα 11: Θέση 3, Φορτίο 0, Σκουπίδια 4
Βήμα 12: Θέση 4, Φορτίο 0, Σκουπίδια 4
Βήμα 13: Θέση 5, Φορτίο 0, Σκουπίδια 4
Βήμα 14: Θέση 6, Φορτίο 0, Σκουπίδια 4
Βήμα 15: Θέση 7, Φορτίο 1, Σκουπίδια 3
Βήμα 16: Θέση 8, Φορτίο 3, Σκουπίδια 1
Βήμα 17: Θέση 1, Φορτίο 0, Σκουπίδια 1
Βήμα 18: Θέση 2, Φορτίο 0, Σκουπίδια 1
Βήμα 19: Θέση 3, Φορτίο 0, Σκουπίδια 1
Βήμα 20: Θέση 4, Φορτίο 0, Σκουπίδια 1
Βήμα 21: Θέση 5, Φορτίο 0, Σκουπίδια 1
Βήμα 22: Θέση 6, Φορτίο 0, Σκουπίδια 1
Βήμα 23: Θέση 7, Φορτίο 0, Σκουπίδια 1
Βήμα 24: Θέση 8, Φορτίο 1, Σκουπίδια 0
Βήμα 25: Θέση 1, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 28
Μέθοδος: Ευριστική (Best-First Search)
```

- Μονοπάτι λύσης: 25 βήματα
- Καταστάσεις που εξετάστηκαν: 28

Σε αυτό το παράδειγμα παρατηρείται μια ενδιαφέρον ιδιαιτερότητα. Και οι τρείς μέθοδοι αναζήτησης καταλήγουν στο ίδιο μονοπάτι λύσης το οποίο αποτελεί και το βέλτιστο, αφού καταλήγει σε αυτό και η BFS. Αυτό που αλλάζει μεταξύ τους είναι ο αριθμός των καταστάσεων που εξετάστηκαν, με την DFS να έχει τον μικρότερο.

Κατάσταση 3

Παίρνουμε την αρχική κατάσταση:

[7, 1, 3, 0, 1, 1, 2, 0, 2, 7, 0]

DFS:

```
_GOAL_FOUND_
[7, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0]

Μονοπάτι λύσης (21 βήματα):
Βήμα 0: Θέση 7, Φορτίο 0, Σκουπίδια 10
Βήμα 1: Θέση 8, Φορτίο 2, Σκουπίδια 8
Βήμα 2: Θέση 7, Φορτίο 2, Σκουπίδια 8
Βήμα 3: Θέση 6, Φορτίο 3, Σκουπίδια 7
Βήμα 4: Θέση 7, Φορτίο 0, Σκουπίδια 7
Βήμα 5: Θέση 6, Φορτίο 1, Σκουπίδια 6
Βήμα 6: Θέση 5, Φορτίο 2, Σκουπίδια 5
Βήμα 7: Θέση 4, Φορτίο 3, Σκουπίδια 4
Βήμα 8: Θέση 7, Φορτίο 0, Σκουπίδια 4
Βήμα 9: Θέση 6, Φορτίο 0, Σκουπίδια 4
Βήμα 10: Θέση 5, Φορτίο 0, Σκουπίδια 4
Βήμα 11: Θέση 4, Φορτίο 0, Σκουπίδια 4
Βήμα 12: Θέση 3, Φορτίο 0, Σκουπίδια 4
Βήμα 13: Θέση 2, Φορτίο 3, Σκουπίδια 1
Βήμα 14: Θέση 7, Φορτίο 0, Σκουπίδια 1
Βήμα 15: Θέση 6, Φορτίο 0, Σκουπίδια 1
Βήμα 16: Θέση 5, Φορτίο 0, Σκουπίδια 1
Βήμα 17: Θέση 4, Φορτίο 0, Σκουπίδια 1
Βήμα 18: Θέση 3, Φορτίο 0, Σκουπίδια 1
Βήμα 19: Θέση 2, Φορτίο 0, Σκουπίδια 1
Βήμα 20: Θέση 1, Φορτίο 1, Σκουπίδια 0
Βήμα 21: Θέση 7, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 29
Μέθοδος: DFS
```

- Μονοπάτι λύσης: 21 βήματα
- Καταστάσεις που εξετάστηκαν: 29

BFS:

```
_GOAL_FOUND_
[7, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0]

Μονοπάτι λύσης (19 βήματα):
Βήμα 0: Θέση 7, Φορτίο 0, Σκουπίδια 10
Βήμα 1: Θέση 6, Φορτίο 2, Σκουπίδια 8
Βήμα 2: Θέση 7, Φορτίο 2, Σκουπίδια 8
Βήμα 3: Θέση 8, Φορτίο 3, Σκουπίδια 7
Βήμα 4: Θέση 7, Φορτίο 0, Σκουπίδια 7
Βήμα 5: Θέση 6, Φορτίο 0, Σκουπίδια 7
Βήμα 6: Θέση 5, Φορτίο 1, Σκουπίδια 6
Βήμα 7: Θέση 4, Φορτίο 2, Σκουπίδια 5
Βήμα 8: Θέση 3, Φορτίο 2, Σκουπίδια 5
Βήμα 9: Θέση 2, Φορτίο 3, Σκουπίδια 4
Βήμα 10: Θέση 7, Φορτίο 0, Σκουπίδια 4
Βήμα 11: Θέση 6, Φορτίο 0, Σκουπίδια 4
Βήμα 12: Θέση 5, Φορτίο 0, Σκουπίδια 4
Βήμα 13: Θέση 4, Φορτίο 0, Σκουπίδια 4
Βήμα 14: Θέση 3, Φορτίο 0, Σκουπίδια 4
Βήμα 15: Θέση 2, Φορτίο 2, Σκουπίδια 2
Βήμα 16: Θέση 1, Φορτίο 3, Σκουπίδια 1
Βήμα 17: Θέση 7, Φορτίο 0, Σκουπίδια 1
Βήμα 18: Θέση 8, Φορτίο 1, Σκουπίδια 0
Βήμα 19: Θέση 7, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 95
Μέθοδος: BFS
```

- Μονοπάτι λύσης: 19 βήματα
- Καταστάσεις που εξετάστηκαν: 95

Best-First Search:

```
_GOAL_FOUND_
[7, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0]

Μονοπάτι λύσης (22 βήματα):
Βήμα 0: Θέση 7, Φορτίο 0, Σκουπίδια 10
Βήμα 1: Θέση 6, Φορτίο 2, Σκουπίδια 8
Βήμα 2: Θέση 5, Φορτίο 3, Σκουπίδια 7
Βήμα 3: Θέση 7, Φορτίο 0, Σκουπίδια 7
Βήμα 4: Θέση 8, Φορτίο 2, Σκουπίδια 5
Βήμα 5: Θέση 7, Φορτίο 2, Σκουπίδια 5
Βήμα 6: Θέση 6, Φορτίο 2, Σκουπίδια 5
Βήμα 7: Θέση 5, Φορτίο 2, Σκουπίδια 5
Βήμα 8: Θέση 4, Φορτίο 3, Σκουπίδια 4
Βήμα 9: Θέση 7, Φορτίο 0, Σκουπίδια 4
Βήμα 10: Θέση 6, Φορτίο 0, Σκουπίδια 4
Βήμα 11: Θέση 5, Φορτίο 0, Σκουπίδια 4
Βήμα 12: Θέση 4, Φορτίο 0, Σκουπίδια 4
Βήμα 13: Θέση 3, Φορτίο 0, Σκουπίδια 4
Βήμα 14: Θέση 2, Φορτίο 3, Σκουπίδια 1
Βήμα 15: Θέση 7, Φορτίο 0, Σκουπίδια 1
Βήμα 16: Θέση 6, Φορτίο 0, Σκουπίδια 1
Βήμα 17: Θέση 5, Φορτίο 0, Σκουπίδια 1
Βήμα 18: Θέση 4, Φορτίο 0, Σκουπίδια 1
Βήμα 19: Θέση 3, Φορτίο 0, Σκουπίδια 1
Βήμα 20: Θέση 2, Φορτίο 0, Σκουπίδια 1
Βήμα 21: Θέση 1, Φορτίο 1, Σκουπίδια 0
Βήμα 22: Θέση 7, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 26
Μέθοδος: Ευριστική (Best-First Search)
```

- Μονοπάτι λύσης: 22 βήματα
- Καταστάσεις που εξετάστηκαν: 26

Σε αυτή την περίπτωση οι τυφλές αναζητήσεις έδωσαν τα αναμενόμενα αποτελέσματα με την DFS να εξετάζει λιγότερες καταστάσεις και την BFS να βρίσκει συντομότερη (βέλτιστη) λύση. Ενδιαφέρον έχει η Best-First αναζήτηση η οποία βρίσκει μεγαλύτερη λύση από την DFS, εξετάζοντας λιγότερες καταστάσεις.

Κατάσταση 4:

Η τελευταία κατάσταση που θα εξετάσουμε είναι η:

[5, 1, 1, 2, 1, 0, 2, 0, 3, 5, 0]

DFS:

```
_GOAL_FOUND_
[5, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0]

Μονοπάτι λύσης (21 βήματα):
Βήμα 0: Θέση 5, Φορτίο 0, Σκουπίδια 10
Βήμα 1: Θέση 6, Φορτίο 2, Σκουπίδια 8
Βήμα 2: Θέση 7, Φορτίο 2, Σκουπίδια 8
Βήμα 3: Θέση 8, Φορτίο 3, Σκουπίδια 7
Βήμα 4: Θέση 5, Φορτίο 0, Σκουπίδια 7
Βήμα 5: Θέση 6, Φορτίο 0, Σκουπίδια 7
Βήμα 6: Θέση 7, Φορτίο 0, Σκουπίδια 7
Βήμα 7: Θέση 8, Φορτίο 2, Σκουπίδια 5
Βήμα 8: Θέση 7, Φορτίο 2, Σκουπίδια 5
Βήμα 9: Θέση 6, Φορτίο 2, Σκουπίδια 5
Βήμα 10: Θέση 5, Φορτίο 2, Σκουπίδια 5
Βήμα 11: Θέση 4, Φορτίο 3, Σκουπίδια 4
Βήμα 12: Θέση 5, Φορτίο 0, Σκουπίδια 4
Βήμα 13: Θέση 4, Φορτίο 0, Σκουπίδια 4
Βήμα 14: Θέση 3, Φορτίο 2, Σκουπίδια 2
Βήμα 15: Θέση 2, Φορτίο 3, Σκουπίδια 1
Βήμα 16: Θέση 5, Φορτίο 0, Σκουπίδια 1
Βήμα 17: Θέση 4, Φορτίο 0, Σκουπίδια 1
Βήμα 18: Θέση 3, Φορτίο 0, Σκουπίδια 1
Βήμα 19: Θέση 2, Φορτίο 0, Σκουπίδια 1
Βήμα 20: Θέση 1, Φορτίο 1, Σκουπίδια 0
Βήμα 21: Θέση 5, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 32
Μέθοδος: DFS
```

- Μονοπάτι λύσης: 21 βήματα
- Καταστάσεις που εξετάστηκαν: 32

BFS:

```
_GOAL_FOUND_
[5, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0]

Μονοπάτι λύσης (17 βήματα):
Βήμα 0: Θέση 5, Φορτίο 0, Σκουπίδια 10
Βήμα 1: Θέση 4, Φορτίο 1, Σκουπίδια 9
Βήμα 2: Θέση 5, Φορτίο 1, Σκουπίδια 9
Βήμα 3: Θέση 6, Φορτίο 3, Σκουπίδια 7
Βήμα 4: Θέση 5, Φορτίο 0, Σκουπίδια 7
Βήμα 5: Θέση 4, Φορτίο 0, Σκουπίδια 7
Βήμα 6: Θέση 3, Φορτίο 2, Σκουπίδια 5
Βήμα 7: Θέση 2, Φορτίο 3, Σκουπίδια 4
Βήμα 8: Θέση 5, Φορτίο 0, Σκουπίδια 4
Βήμα 9: Θέση 6, Φορτίο 0, Σκουπίδια 4
Βήμα 10: Θέση 7, Φορτίο 0, Σκουπίδια 4
Βήμα 11: Θέση 8, Φορτίο 3, Σκουπίδια 1
Βήμα 12: Θέση 5, Φορτίο 0, Σκουπίδια 1
Βήμα 13: Θέση 4, Φορτίο 0, Σκουπίδια 1
Βήμα 14: Θέση 3, Φορτίο 0, Σκουπίδια 1
Βήμα 15: Θέση 2, Φορτίο 0, Σκουπίδια 1
Βήμα 16: Θέση 1, Φορτίο 1, Σκουπίδια 0
Βήμα 17: Θέση 5, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 109
Μέθοδος: BFS
```

- Μονοπάτι λύσης: 17 βήματα
- Καταστάσεις που εξετάστηκαν: 109

Best-First Search:

```
_GOAL_FOUND_
[5, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0]

Μονοπάτι λύσης (17 βήματα):
Βήμα 0: Θέση 5, Φορτίο 0, Σκουπίδια 10
Βήμα 1: Θέση 6, Φορτίο 2, Σκουπίδια 8
Βήμα 2: Θέση 5, Φορτίο 2, Σκουπίδια 8
Βήμα 3: Θέση 4, Φορτίο 3, Σκουπίδια 7
Βήμα 4: Θέση 5, Φορτίο 0, Σκουπίδια 7
Βήμα 5: Θέση 4, Φορτίο 0, Σκουπίδια 7
Βήμα 6: Θέση 3, Φορτίο 2, Σκουπίδια 5
Βήμα 7: Θέση 2, Φορτίο 3, Σκουπίδια 4
Βήμα 8: Θέση 5, Φορτίο 0, Σκουπίδια 4
Βήμα 9: Θέση 6, Φορτίο 0, Σκουπίδια 4
Βήμα 10: Θέση 7, Φορτίο 0, Σκουπίδια 4
Βήμα 11: Θέση 8, Φορτίο 3, Σκουπίδια 1
Βήμα 12: Θέση 5, Φορτίο 0, Σκουπίδια 1
Βήμα 13: Θέση 4, Φορτίο 0, Σκουπίδια 1
Βήμα 14: Θέση 3, Φορτίο 0, Σκουπίδια 1
Βήμα 15: Θέση 2, Φορτίο 0, Σκουπίδια 1
Βήμα 16: Θέση 1, Φορτίο 1, Σκουπίδια 0
Βήμα 17: Θέση 5, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 23
Μέθοδος: Ευριστική (Best-First Search)
```

- Μονοπάτι λύσης: 17 βήματα
- Καταστάσεις που εξετάστηκαν: 23

Για αυτή την κατάσταση, η Best-First έδωσε τα καλύτερα αποτελέσματα, βρίσκοντας την βέλτιστη λύση που βρήκε και η BFS και εξετάζοντας λιγότερες καταστάσεις ακόμα και από την DFS.

Συμπεράσματα

Έχοντας εξετάσει τις αρχικές καταστάσεις:

1. **[3, 2, 3, 0, 0, 2, 0, 1, 8, 3, 0]** (σελ. 9, 10, 17)
2. **[4, 1, 0, 2, 0, 3, 3, 0, 1, 4, 0]** (σελ. 20, 21)
3. **[1, 0, 3, 0, 1, 1, 0, 2, 3, 1, 0]** (σελ. 22, 23)
4. **[7, 1, 3, 0, 1, 1, 2, 0, 2, 7, 0]** (σελ. 24, 25)
5. **[5, 1, 1, 2, 1, 0, 2, 0, 3, 5, 0]** (σελ. 26, 27)

έχουμε πετύχει μια ολοκληρωμένη παρουσίαση της συμπεριφοράς των τριών μεθόδων αναζήτησης DFS, BFS και Best-First Search.

Γενικά περιμέναμε πως η DFS δεν θα βρίσκει την βέλτιστη λύση αλλά θα κάνει την μικρότερη αναζήτηση, η BFS βρίσκοντας πάντα την βέλτιστη θα πραγματοποιεί τις μεγαλύτερες αναζητήσεις, και η ευριστική Best-First Search θα βρίσκει συνήθως καλύτερη λύση από την DFS εκτελώντας τις μικρότερες αναζητήσεις.

Στην πράξη όμως είδαμε πως τα αποτελέσματα επηρεάζονται σε πολύ μεγάλο βαθμό από τις αρχικές καταστάσεις που χρησιμοποιούμε. Πέραν των περιπτώσεων με τα αναμενόμενα αποτελέσματα, είδαμε και περιπτώσεις όπου και οι τρείς αναζητήσεις καταλήγουν στο βέλτιστο μονοπάτι, η Best-First βρίσκει λύση με μεγαλύτερο μονοπάτι από την DFS όπως επίσης η Best-First να βρίσκει το βέλτιστο μονοπάτι εξετάζοντας όμως λιγότερες καταστάσεις από την DFS.

Συνοψίζοντας, την μεγαλύτερη διαφορά στα αποτελέσματα την προκαλούν οι αρχικές καταστάσεις. Ακόμα κι έτσι όμως, βλέπουμε πως η Best-First search έχοντας σαν μοναδική πληροφορία τα εναπομείναντα σκουπίδια του χώρου, αδυνατεί να μας δίνει σταθερά καλύτερα αποτελέσματα από τις τυφλές μεθόδους.