Τεχνητή Νοημοσύνη

Ρόμπερτ Πολοβίνα – 23390338

Εξάμηνο 7ο

Απαλλακτική Εργασία Εξαμήνου – Ρομποτική Σκούπα

Έγγραφο Τεκμηρίωσης

2025-2026

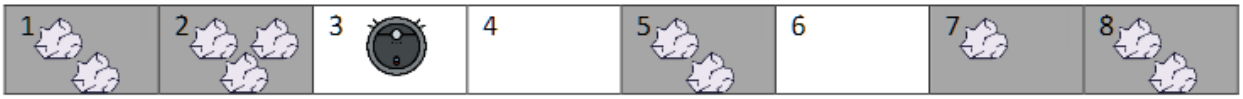
Περιεχόμενα

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]

Τελεστές Μετάβασης, Απόγονοι και Στόχος

Οι **τελεστές μετάβασης** είναι οι λειτουργίες της σκούπας. Μαζί με την συνάρτηση *find_children(),* καθορίζουν την μετάβαση από μία κατάσταση του χώρου καταστάσεων στην επόμενη και η συνάρτηση *is_goal_state()* είναι αυτή που θα αποφανθεί αν η τρέχουσα κατάσταση είναι και η τελική κατάσταση που ψάχνουμε. Οι κώδικες των τελεστών είναι βασισμένοι στους κώδικες που έχουν δοθεί στα έγγραφα του μαθήματος. Συγκεκριμένα:

- **move_left(state):** μετακινεί την σκούπα ένα πλακάκι αριστερά
- **move_right(state):** μετακινεί την σκούπα ένα πλακάκι δεξιά
- **empty_robot(state):** η σκούπα αδειάζει το φορτίο της όταν βρίσκεται στην βάση της
- **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 συναρτήσεις ***move_right()*** και ***move_left()***:

1. Ελέγχουν τα όρια του διαδρόμου
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
    if 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
    if state[0]< 8:
        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[state[0]]
                state[state[0]]= 0

    return state
```

- Η συνάρτηση ***empty_robot()***:
    1. Ελέγχει αν η σκούπα βρίσκεται στην βάση της και
    2. αν έχει φορτίο το μηδενίζει, αλλιώς δεν μεταβάλλεται η κατάσταση

```
#Άδειασμα σκουπιδιών
def empty_robot(state):
    #αν η σκούπα βρίσκεται στη βάση και έχει φορτίο
    if state[0]== state[-2] and state[-1]> 0:
        state[-1]= 0
    return state
```

- 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.

```
#Έλεγχος 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 empty_robot()
    empty_state= copy.deepcopy(state)
    empty_child= empty_robot(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 προστέθει 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 (28 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 6, Φορτίο 3, Σκουπίδια 7
Βήμα 6: Θέση 5, Φορτίο 3, Σκουπίδια 7
Βήμα 7: Θέση 4, Φορτίο 3, Σκουπίδια 7
Βήμα 8: Θέση 3, Φορτίο 3, Σκουπίδια 7
Βήμα 9: Θέση 3, Φορτίο 0, Σκουπίδια 7
Βήμα 10: Θέση 4, Φορτίο 0, Σκουπίδια 7
Βήμα 11: Θέση 5, Φορτίο 0, Σκουπίδια 7
Βήμα 12: Θέση 6, Φορτίο 0, Σκουπίδια 7
Βήμα 13: Θέση 7, Φορτίο 0, Σκουπίδια 7
Βήμα 14: Θέση 8, Φορτίο 2, Σκουπίδια 5
Βήμα 15: Θέση 7, Φορτίο 2, Σκουπίδια 5
Βήμα 16: Θέση 6, Φορτίο 2, Σκουπίδια 5
Βήμα 17: Θέση 5, Φορτίο 2, Σκουπίδια 5
Βήμα 18: Θέση 4, Φορτίο 2, Σκουπίδια 5
Βήμα 19: Θέση 3, Φορτίο 2, Σκουπίδια 5
Βήμα 20: Θέση 3, Φορτίο 0, Σκουπίδια 5
Βήμα 21: Θέση 2, Φορτίο 3, Σκουπίδια 2
Βήμα 22: Θέση 3, Φορτίο 3, Σκουπίδια 2
Βήμα 23: Θέση 3, Φορτίο 0, Σκουπίδια 2
Βήμα 24: Θέση 2, Φορτίο 0, Σκουπίδια 2
Βήμα 25: Θέση 1, Φορτίο 2, Σκουπίδια 0
Βήμα 26: Θέση 2, Φορτίο 2, Σκουπίδια 0
Βήμα 27: Θέση 3, Φορτίο 2, Σκουπίδια 0
Βήμα 28: 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 28 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 (24 βήματα):
Βήμα 0: Θέση 3, Φορτίο 0, Σκουπίδια 10
Βήμα 1: Θέση 2, Φορτίο 3, Σκουπίδια 7
Βήμα 2: Θέση 3, Φορτίο 3, Σκουπίδια 7
Βήμα 3: Θέση 3, Φορτίο 0, Σκουπίδια 7
Βήμα 4: Θέση 2, Φορτίο 0, Σκουπίδια 7
Βήμα 5: Θέση 1, Φορτίο 2, Σκουπίδια 5
Βήμα 6: Θέση 2, Φορτίο 2, Σκουπίδια 5
Βήμα 7: Θέση 3, Φορτίο 2, Σκουπίδια 5
Βήμα 8: Θέση 3, Φορτίο 0, Σκουπίδια 5
Βήμα 9: Θέση 4, Φορτίο 0, Σκουπίδια 5
Βήμα 10: Θέση 5, Φορτίο 2, Σκουπίδια 3
Βήμα 11: Θέση 4, Φορτίο 2, Σκουπίδια 3
Βήμα 12: Θέση 3, Φορτίο 2, Σκουπίδια 3
Βήμα 13: Θέση 3, Φορτίο 0, Σκουπίδια 3
Βήμα 14: Θέση 4, Φορτίο 0, Σκουπίδια 3
Βήμα 15: Θέση 5, Φορτίο 0, Σκουπίδια 3
Βήμα 16: Θέση 6, Φορτίο 0, Σκουπίδια 3
Βήμα 17: Θέση 7, Φορτίο 1, Σκουπίδια 2
Βήμα 18: Θέση 8, Φορτίο 3, Σκουπίδια 0
Βήμα 19: Θέση 7, Φορτίο 3, Σκουπίδια 0
Βήμα 20: Θέση 6, Φορτίο 3, Σκουπίδια 0
Βήμα 21: Θέση 5, Φορτίο 3, Σκουπίδια 0
Βήμα 22: Θέση 4, Φορτίο 3, Σκουπίδια 0
Βήμα 23: Θέση 3, Φορτίο 3, Σκουπίδια 0
Βήμα 24: Θέση 3, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 308
Μέθοδος: BFS
```

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

- ο BFS εξέτασε πάρα πολλές παραπάνω καταστάσεις σε σχέση με τον DFS (308 έναντι 39)
- Βρήκε την βέλτιστη διαδρομή κατά 4 λιγότερα βήματα (24 έναντι 28).

Δέντρα Αναζήτησης DFS & BFS

Εδώ παρουσιάζεται σχηματικά, μετά την επεξήγηση, ο τρόπος λειτουργίας των δύο αλγορίθμων. Για να μην προχωρήσουμε σε μεγάλο βάθος δέντρου, έστω η απλοποιημένη αρχική κατάσταση:

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

με στόχο:

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

Επίπεδο 1:

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

S0 (δεν αλλάζει): *empty_robot* -> [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): *empty_robot* -> [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:

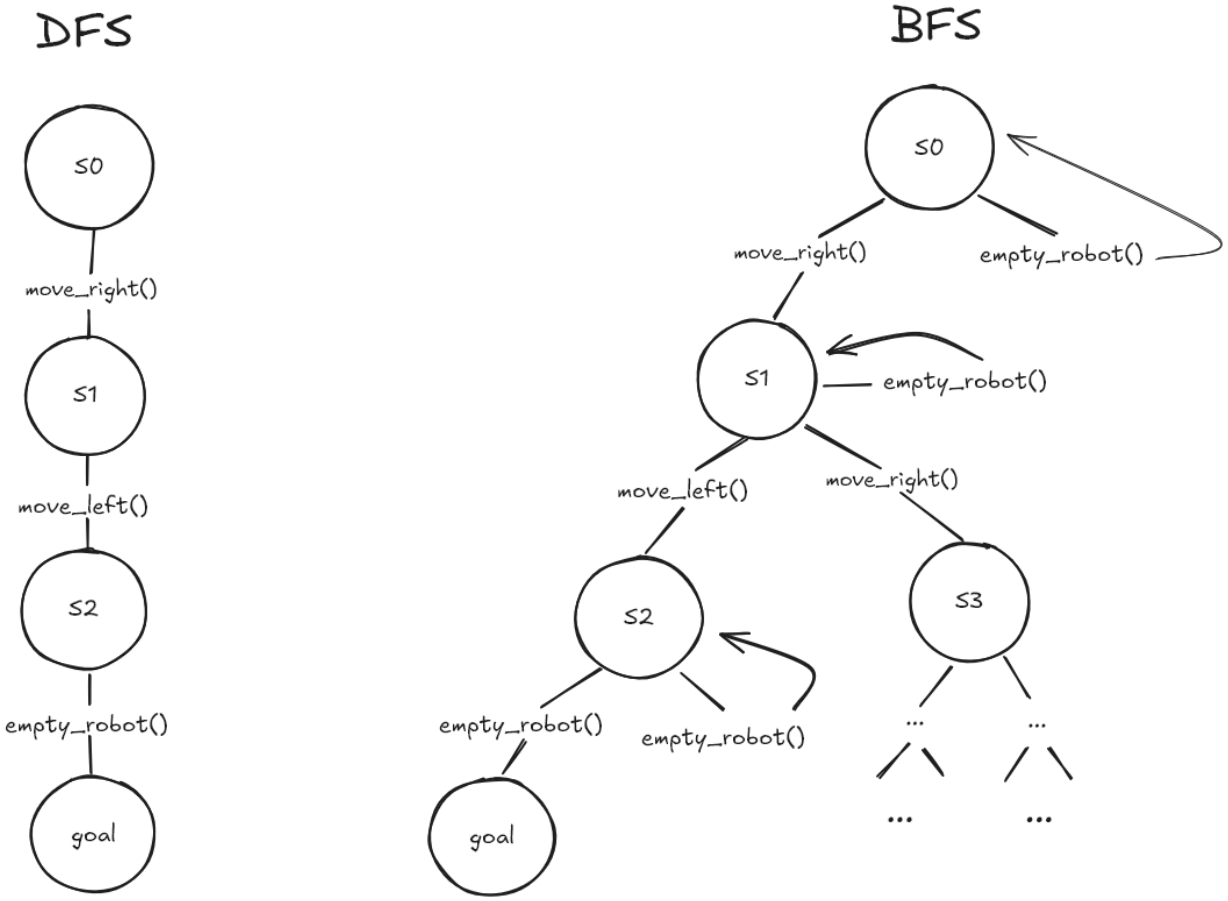
Η σκούπα βρίσκεται τώρα στην βάση της με φορτίο 1 σκουπίδι. Συνεπώς θα εκτελέσει την λειτουργία ***empty_robot*** και θα βρεθεί στην τελική κατάσταση, λύνοντας το πρόβλημα.

goal: *empty_robot* -> [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

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.

Η διαφορά αυτή απεικονίζεται σχηματικά μέσω της πληρότητας του δέντρου της BFS. Η DFS εφόσον βρήκε τον στόχο από την πρώτη διαδρομή που εξέτασε δεν θα γυρίσει πίσω για να δει τις υπόλοιπες επιλογές που είδε η BFS.



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 και η απόσταση της σκούπας από την βάση. Τα δύο κριτήρια δηλαδή, που αποτελούν τον τελικό στόχο του προβλήματος.

```
def heuristic(state):
    garbage_sum= sum(state[1:9])
    distance_to_base= abs(state[0]- state[-2])

    return garbage_sum+ distance_to_base
```

Πρακτικά ενώ οι DFS και BFS βασίζονται στην σειρά των καταστάσεων μέσα σε δύο ξεχωριστές δομές (**front** και **queue)**, η ευριστική αναζήτηση βασίζεται στην προτεραιότητα τους. Χρησιμοποιήθηκε η ευριστική μέθοδος **Best-First Search** η οποία χρειάζεται μία ενιαία δομή που θα πρέπει να κρατάει και να ταξινομεί 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 για την υλοποίηση της Best-First Search έγινε χρήση της βιβλιοθήκης *heapq,* η οποία χρησιμοποιείται για την ασφαλή και αποδοτικότερη υλοποίηση δομών ουράς προτεραιότητας λόγω του τρόπου της διαχείρισης των προτεραιοτήτων.

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



    while queue:
      #Αφαίρεση της κατάστασης με την μεγαλύτερη προτεραιότητα από την ουρά και
```

```
      #ανάθεση της στην μεταβλητή path και της ευριστικής τιμής της στην h_value
        h_value, path= heapq.heappop(queue)
        current= path[-1] #path[-1] -> η πιο πρόσφατη κατάσταση (τρέχουσα)
        path_new= path
        states_examined+= 1

        if current in closed:
            continue

        closed.append(current)

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

        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 προτεραιότητας
                heapq.heappush(queue, (h_child, path_new))

        #Ο έλεγχος του στόχου γίνεται εδώ και όχι πριν την δημιουργία απογόνων
        #γιατί για να τυπώσω το μονοπάτι της λύσης χρειάζομαι το 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 ουρά προτεραιότητας (min-heap) 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. Στην εύρεση απογόνων βρίσκουμε επίσης και την ευριστική τιμή του παιδιού, το οποίο έπειτα 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*** η οποία στο τέλος της επανάληψης θα περιέχει ολόκληρο το μονοπάτι. Δεν μπορούμε να χρησιμοποιήσουμε την ***path*** διότι περιέχει μόνο 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).

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

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

Συνολικές καταστάσεις που εξετάστηκαν: 42
Μέθοδος: Ευριστική (Best-First Search)
```

Από την εκτέλεση του προγράμματος, βλέπουμε ότι η ευριστική αναζήτηση βρήκε το βέλτιστο μονοπάτι που βρήκε και η BFS (24 βήματα) εξετάζοντας όμως πάρα πολύ λιγότερες κατάστασεις (42 έναντι 308, μόλις 3 περισσότερες από την DFS).

Παρόλο που στην πράξη η ευριστική αναζήτηση δουλεύει διαφορετικά, όσον αφορά το αποτέλεσμα είναι σαν να «συνδυάζουμε» την βελτιστοποιημένη διαδρομή της BFS με την γρήγορη και αποδοτική αναζήτηση της DFS.

Να σημειωθεί όμως ότι η Best-First Search δεν εγγυάται γενικά την εύρεση της συντομότερης 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]

    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

Για την σύγκριση των τριών αλγορίθμων αναζήτησης θα χρησιμοποιηθούν τρεις διαφορετικές αρχικές καταστάσεις.

Κατάσταση 1η

Αρχικά θα δούμε την απλή αρχική κατάσταση που χρησιμοποιήθηκε στην ενότητα της σχηματικής απεικόνησης των δέντρων (σελ. 11) για να επαληθεύσουμε τον τρόπο λειτουργίας των αλγορίθμων:

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

DFS:

```
____BEGIN__SEARCHING____

Κατάσταση που εξετάζεται:
[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0]

Μέτωπο:
  1: [1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0]

Ουρά μονοπατιών:
  1: [[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0]]

====================================

Κατάσταση που εξετάζεται:
[2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]

Μέτωπο:
  1: [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
  2: [1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0]

Ουρά μονοπατιών:
  1: [[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0], [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]]

  2: [[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0], [1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0]]

====================================
```

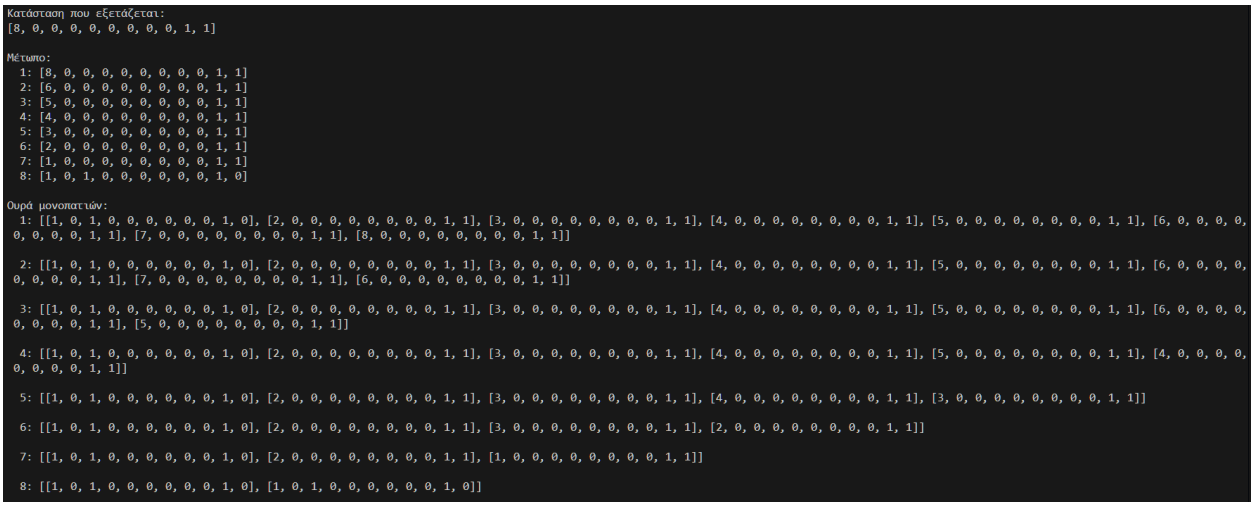
Χάρις στη παράλληλη παρακολούθηση του μετώπου και της ουράς βλέπουμε ότι η περιγράφη στην ενότητα των δέντρων (σελ. 11) συνάδει με το στιγμιότυπο της εκτέλεσης. Η σκούπα βλέπει ότι είτε θα αδειάσει (**empty_robot()**) και θα βρέθει στην ίδια κατάσταση που βρίσκεται ήδη ή θα μετακινηθεί μόνο δεξιά (**move_right()**) διότι βρίσκεται στο αριστερό άκρο.

```
Κατάσταση που εξετάζεται:
[3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]

Μέτωπο:
  1: [3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
  2: [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
  3: [1, 0, 1, 0, 0, 0, 0, 0, 1, 0]

Ουρά μονοπατιών:
  1: [[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0], [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1], [3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]]

  2: [[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0], [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]]

  3: [[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0], [1, 0, 1, 0, 0, 0, 0, 0, 1, 0]]
```

```
Κατάσταση που εξετάζεται:
[4, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]

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

Ουρά μονοπατιών:
  1: [[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0], [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1], [3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1], [4, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]]

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

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

  4: [[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0], [1, 0, 1, 0, 0, 0, 0, 0, 1, 0]]
```

```
Κατάσταση που εξετάζεται:
[5, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]

Μέτωπο:
  1: [5, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
  2: [3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
  3: [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
  4: [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
  5: [1, 0, 1, 0, 0, 0, 0, 0, 1, 0]

Ουρά μονοπατιών:
  1: [[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0], [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1], [3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1], [4, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1], [5, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]]

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

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

  4: [[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0], [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]]

  5: [[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0], [1, 0, 1, 0, 0, 0, 0, 0, 1, 0]]
```

Στο επόμενο στάδιο της εκτέλεσης φαίνεται το μειονέκτημα του DFS που είχαμε περιγράψει και στην σελίδα 11. Βλέπουμε ότι ο DFS προχωράει σε εμβάθυνση στο μονοπάτι της περιττής κατάστασης [3, 0…0, 1, 1] η οποία δεν οδηγεί στον στόχο ενώ η σωστή διαδρομή έχει κατέβει στο τέλος της ουράς περιμένοντας την σειρά της.



Ο DFS συνεχίζει να εξετάζει το λάθος μονοπάτι έως ότου να χτυπήσει το τέλος του, το δεξί άκρο (θέση 8). Όταν γίνει αυτό, αφαιρεί όλες τις καταστάσεις αυτού του μονοπατιού από το μέτωπο και προχωράει στην εξέταση του σωστού μονοπατιού και τελικά στην εύρεση του στόχου.

```
Κατάσταση που εξετάζεται:
[1, 0, 0, 0, 0, 0, 0, 0, 1, 1]

Μέτωπο:
  1: [1, 0, 0, 0, 0, 0, 0, 0, 1, 1]
  2: [1, 0, 1, 0, 0, 0, 0, 0, 1, 0]

Ουρά μονοπατιών:
  1: [[1, 0, 1, 0, 0, 0, 0, 0, 1, 0], [2, 0, 0, 0, 0, 0, 0, 0, 1, 1], [1, 0, 0, 0, 0, 0, 0, 0, 1, 1]]

  2: [[1, 0, 1, 0, 0, 0, 0, 0, 1, 0], [1, 0, 1, 0, 0, 0, 0, 0, 1, 0]]

======================================

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

Μονοπάτι λύσης (3 βήματα):
Βήμα 0: Θέση 1, Φορτίο 0, Σκουπίδια 1
Βήμα 1: Θέση 2, Φορτίο 1, Σκουπίδια 0
Βήμα 2: Θέση 1, Φορτίο 1, Σκουπίδια 0
Βήμα 3: Θέση 1, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 9
Μέθοδος: DFS
```

BFS:

```
____BEGIN__SEARCHING____

Κατάσταση που εξετάζεται:
[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0]

Μέτωπο:
  1: [1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0]

Ουρά μονοπατιών:
  1: [[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0]]

====================================

Κατάσταση που εξετάζεται:
[2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]

Μέτωπο:
  1: [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
  2: [1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0]

Ουρά μονοπατιών:
  1: [[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0], [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]]

  2: [[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0], [1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0]]
```

Ο αλγόριθμος BFS ξεκινάει και αυτός με τις δύο (και μόνες) πιθανές ενέργειες που είδε και ο DFS.

```
Κατάσταση που εξετάζεται:
[1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]

Μέτωπο:
  1: [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
  2: [3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
  3: [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]

Ουρά μονοπατιών:
  1: [[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0], [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]]

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

  3: [[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0], [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1], [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]]
```

Επειδή όμως βλέπει όλες τις καταστάσεις ενός επιπέδου πριν προχωρήσει στο επόμενο, ψάχνει και βλέπει περισσότερες καταστάσεις ταυτόχρονα σε σχέση με τον DFS.

```
Κατάσταση που εξετάζεται:
[3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]

Μέτωπο:
  1: [3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
  2: [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
  3: [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
  4: [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
  5: [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

Ουρά μονοπατιών:
  1: [[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0], [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1], [3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]]

  2: [[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0], [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1], [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]]

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

  4: [[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0], [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1], [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]]

  5: [[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0], [2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]]

====================================

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

Μονοπάτι λύσης (3 βήματα):
Βήμα 0: Θέση 1, Φορτίο 0, Σκουπίδια 1
Βήμα 1: Θέση 2, Φορτίο 1, Σκουπίδια 0
Βήμα 2: Θέση 1, Φορτίο 1, Σκουπίδια 0
Βήμα 3: Θέση 1, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 4
Μέθοδος: BFS
```

Έτσι εντοπίζει τον στόχο χωρίς να χρειάζεται να σπαταλήσει ενέργειες για να ψάξει το λάθος μονοπάτι μέχρι το τέλος του. Για τον λόγο αυτό σε συνδυασμό με την υπερβολικά απλή αρχική κατάσταση που έχουμε, ο BFS εξετάζει λιγότερα βήματα από τον DFS ενώ στις πιο περίπλοκες καταστάσεις (περισσότερα σκουπίδια) θα συμβαίνει το αντίθετο.

Ευριστική:

Στην συγκεκριμένη αρχική κατάσταση που εξετάζουμε, η ευριστική αναζήτηση δεν μπορεί να προσφέρει κάποιο πλεονέκτημα.

```
____BEGIN__SEARCHING____

Κατάσταση που εξετάζεται:
[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0]
Ευριστική τιμή h= 1

Κατάσταση που εξετάζεται:
[2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
Ευριστική τιμή h= 1

Κατάσταση που εξετάζεται:
[1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
Ευριστική τιμή h= 0

Κατάσταση που εξετάζεται:
[1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
Ευριστική τιμή h= 0

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

Μονοπάτι λύσης (3 βήματα):
Βήμα 0: Θέση 1, Φορτίο 0, Σκουπίδια 1
Βήμα 1: Θέση 2, Φορτίο 1, Σκουπίδια 0
Βήμα 2: Θέση 1, Φορτίο 1, Σκουπίδια 0
Βήμα 3: Θέση 1, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 4
Μέθοδος: Ευριστική (Best-First Search)
```

Κατάσταση 2η

Συνεχίζουμε την ανάλυση μας με την παρακάτω αρχική κατάσταση:

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

Έχοντας επαληθεύσει την ορθότητα των αλγορίθμων στην προηγούμενη κατάσταση και λόγω του πολύ μεγάλου αριθμού των καταστάσεων που τυπώνονται μαζί με τις ουρές και τα μέτωπα (ειδικά στην επίλυση με BFS), θα παρουσιάζονται και θα σχολιάζονται στιγμιότυπα μόνο του μονοπατιού της λύσης.

DFS:

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

Μονοπάτι λύσης (27 βήματα):
Βήμα 0: Θέση 4, Φορτίο 0, Σκουπίδια 10
Βήμα 1: Θέση 5, Φορτίο 3, Σκουπίδια 7
Βήμα 2: Θέση 4, Φορτίο 3, Σκουπίδια 7
Βήμα 3: Θέση 4, Φορτίο 0, Σκουπίδια 7
Βήμα 4: Θέση 5, Φορτίο 0, Σκουπίδια 7
Βήμα 5: Θέση 6, Φορτίο 3, Σκουπίδια 4
Βήμα 6: Θέση 5, Φορτίο 3, Σκουπίδια 4
Βήμα 7: Θέση 4, Φορτίο 3, Σκουπίδια 4
Βήμα 8: Θέση 4, Φορτίο 0, Σκουπίδια 4
Βήμα 9: Θέση 5, Φορτίο 0, Σκουπίδια 4
Βήμα 10: Θέση 6, Φορτίο 0, Σκουπίδια 4
Βήμα 11: Θέση 7, Φορτίο 0, Σκουπίδια 4
Βήμα 12: Θέση 8, Φορτίο 1, Σκουπίδια 3
Βήμα 13: Θέση 7, Φορτίο 1, Σκουπίδια 3
Βήμα 14: Θέση 6, Φορτίο 1, Σκουπίδια 3
Βήμα 15: Θέση 5, Φορτίο 1, Σκουπίδια 3
Βήμα 16: Θέση 4, Φορτίο 1, Σκουπίδια 3
Βήμα 17: Θέση 4, Φορτίο 0, Σκουπίδια 3
Βήμα 18: Θέση 3, Φορτίο 2, Σκουπίδια 1
Βήμα 19: Θέση 4, Φορτίο 2, Σκουπίδια 1
Βήμα 20: Θέση 4, Φορτίο 0, Σκουπίδια 1
Βήμα 21: Θέση 3, Φορτίο 0, Σκουπίδια 1
Βήμα 22: Θέση 2, Φορτίο 0, Σκουπίδια 1
Βήμα 23: Θέση 1, Φορτίο 1, Σκουπίδια 0
Βήμα 24: Θέση 2, Φορτίο 1, Σκουπίδια 0
Βήμα 25: Θέση 3, Φορτίο 1, Σκουπίδια 0
Βήμα 26: Θέση 4, Φορτίο 1, Σκουπίδια 0
Βήμα 27: Θέση 4, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 40
Μέθοδος: DFS
```

BFS:

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

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

Συνολικές καταστάσεις που εξετάστηκαν: 265
Μέθοδος: BFS
```

Ευριστική:

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

Μονοπάτι λύσης (27 βήματα):
Βήμα 0: Θέση 4, Φορτίο 0, Σκουπίδια 10
Βήμα 1: Θέση 5, Φορτίο 3, Σκουπίδια 7
Βήμα 2: Θέση 4, Φορτίο 3, Σκουπίδια 7
Βήμα 3: Θέση 4, Φορτίο 0, Σκουπίδια 7
Βήμα 4: Θέση 3, Φορτίο 2, Σκουπίδια 5
Βήμα 5: Θέση 4, Φορτίο 2, Σκουπίδια 5
Βήμα 6: Θέση 4, Φορτίο 0, Σκουπίδια 5
Βήμα 7: Θέση 5, Φορτίο 0, Σκουπίδια 5
Βήμα 8: Θέση 6, Φορτίο 3, Σκουπίδια 2
Βήμα 9: Θέση 5, Φορτίο 3, Σκουπίδια 2
Βήμα 10: Θέση 4, Φορτίο 3, Σκουπίδια 2
Βήμα 11: Θέση 4, Φορτίο 0, Σκουπίδια 2
Βήμα 12: Θέση 3, Φορτίο 0, Σκουπίδια 2
Βήμα 13: Θέση 2, Φορτίο 0, Σκουπίδια 2
Βήμα 14: Θέση 1, Φορτίο 1, Σκουπίδια 1
Βήμα 15: Θέση 2, Φορτίο 1, Σκουπίδια 1
Βήμα 16: Θέση 3, Φορτίο 1, Σκουπίδια 1
Βήμα 17: Θέση 4, Φορτίο 1, Σκουπίδια 1
Βήμα 18: Θέση 4, Φορτίο 0, Σκουπίδια 1
Βήμα 19: Θέση 5, Φορτίο 0, Σκουπίδια 1
Βήμα 20: Θέση 6, Φορτίο 0, Σκουπίδια 1
Βήμα 21: Θέση 7, Φορτίο 0, Σκουπίδια 1
Βήμα 22: Θέση 8, Φορτίο 1, Σκουπίδια 0
Βήμα 23: Θέση 7, Φορτίο 1, Σκουπίδια 0
Βήμα 24: Θέση 6, Φορτίο 1, Σκουπίδια 0
Βήμα 25: Θέση 5, Φορτίο 1, Σκουπίδια 0
Βήμα 26: Θέση 4, Φορτίο 1, Σκουπίδια 0
Βήμα 27: Θέση 4, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 37
Μέθοδος: Ευριστική (Best-First Search)
```

Ο DFS εκτελεί την αναζήτηση εξετάζοντας πάρα πολλές λιγότερες καταστάσεις από τον BFS ο οποίος όμως βρίσκει την βέλτιστη λύση κατά τρία λιγότερα βήματα. Η ευριστική μέθοδος βρίσκει τα ίδια βήματα επίλυσης με τον DFS εξετάζοντας τρεις καταστάσεις λιγότερο.

Τα αποτελέσματα αυτά δεν είναι κάτι το μη αναμενόμενο.

Κατάσταση 3η

Τέλος, θα εξετάσουμε την παρακάτω 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 (46 βήματα):
Βήμα 0: Θέση 1, Φορτίο 0, Σκουπίδια 10
Βήμα 1: Θέση 2, Φορτίο 3, Σκουπίδια 7
Βήμα 2: Θέση 1, Φορτίο 3, Σκουπίδια 7
Βήμα 3: Θέση 1, Φορτίο 0, Σκουπίδια 7
Βήμα 4: Θέση 2, Φορτίο 0, Σκουπίδια 7
Βήμα 5: Θέση 3, Φορτίο 0, Σκουπίδια 7
Βήμα 6: Θέση 4, Φορτίο 1, Σκουπίδια 6
Βήμα 7: Θέση 5, Φορτίο 2, Σκουπίδια 5
Βήμα 8: Θέση 6, Φορτίο 2, Σκουπίδια 5
Βήμα 9: Θέση 7, Φορτίο 3, Σκουπίδια 4
Βήμα 10: Θέση 6, Φορτίο 3, Σκουπίδια 4
Βήμα 11: Θέση 5, Φορτίο 3, Σκουπίδια 4
Βήμα 12: Θέση 4, Φορτίο 3, Σκουπίδια 4
Βήμα 13: Θέση 3, Φορτίο 3, Σκουπίδια 4
Βήμα 14: Θέση 2, Φορτίο 3, Σκουπίδια 4
Βήμα 15: Θέση 1, Φορτίο 3, Σκουπίδια 4
Βήμα 16: Θέση 1, Φορτίο 0, Σκουπίδια 4
Βήμα 17: Θέση 2, Φορτίο 0, Σκουπίδια 4
Βήμα 18: Θέση 3, Φορτίο 0, Σκουπίδια 4
Βήμα 19: Θέση 4, Φορτίο 0, Σκουπίδια 4
Βήμα 20: Θέση 5, Φορτίο 0, Σκουπίδια 4
Βήμα 21: Θέση 6, Φορτίο 0, Σκουπίδια 4
Βήμα 22: Θέση 7, Φορτίο 1, Σκουπίδια 3
Βήμα 23: Θέση 8, Φορτίο 3, Σκουπίδια 1
Βήμα 24: Θέση 7, Φορτίο 3, Σκουπίδια 1
Βήμα 25: Θέση 6, Φορτίο 3, Σκουπίδια 1
Βήμα 26: Θέση 5, Φορτίο 3, Σκουπίδια 1
Βήμα 27: Θέση 4, Φορτίο 3, Σκουπίδια 1
Βήμα 28: Θέση 3, Φορτίο 3, Σκουπίδια 1
Βήμα 29: Θέση 2, Φορτίο 3, Σκουπίδια 1
Βήμα 30: Θέση 1, Φορτίο 3, Σκουπίδια 1
Βήμα 31: Θέση 1, Φορτίο 0, Σκουπίδια 1
Βήμα 32: Θέση 2, Φορτίο 0, Σκουπίδια 1
Βήμα 33: Θέση 3, Φορτίο 0, Σκουπίδια 1
Βήμα 34: Θέση 4, Φορτίο 0, Σκουπίδια 1
Βήμα 35: Θέση 5, Φορτίο 0, Σκουπίδια 1
Βήμα 36: Θέση 6, Φορτίο 0, Σκουπίδια 1
Βήμα 37: Θέση 7, Φορτίο 0, Σκουπίδια 1
Βήμα 38: Θέση 8, Φορτίο 1, Σκουπίδια 0
Βήμα 39: Θέση 7, Φορτίο 1, Σκουπίδια 0
Βήμα 40: Θέση 6, Φορτίο 1, Σκουπίδια 0
Βήμα 41: Θέση 5, Φορτίο 1, Σκουπίδια 0
Βήμα 42: Θέση 4, Φορτίο 1, Σκουπίδια 0
Βήμα 43: Θέση 3, Φορτίο 1, Σκουπίδια 0
Βήμα 44: Θέση 2, Φορτίο 1, Σκουπίδια 0
Βήμα 45: Θέση 1, Φορτίο 1, Σκουπίδια 0
Βήμα 46: Θέση 1, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 53
Μέθοδος: DFS
```

BFS:

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

Μονοπάτι λύσης (38 βήματα):
Βήμα 0: Θέση 1, Φορτίο 0, Σκουπίδια 10
Βήμα 1: Θέση 2, Φορτίο 3, Σκουπίδια 7
Βήμα 2: Θέση 1, Φορτίο 3, Σκουπίδια 7
Βήμα 3: Θέση 1, Φορτίο 0, Σκουπίδια 7
Βήμα 4: Θέση 2, Φορτίο 0, Σκουπίδια 7
Βήμα 5: Θέση 3, Φορτίο 0, Σκουπίδια 7
Βήμα 6: Θέση 4, Φορτίο 1, Σκουπίδια 6
Βήμα 7: Θέση 3, Φορτίο 1, Σκουπίδια 6
Βήμα 8: Θέση 2, Φορτίο 1, Σκουπίδια 6
Βήμα 9: Θέση 1, Φορτίο 1, Σκουπίδια 6
Βήμα 10: Θέση 1, Φορτίο 0, Σκουπίδια 6
Βήμα 11: Θέση 2, Φορτίο 0, Σκουπίδια 6
Βήμα 12: Θέση 3, Φορτίο 0, Σκουπίδια 6
Βήμα 13: Θέση 4, Φορτίο 0, Σκουπίδια 6
Βήμα 14: Θέση 5, Φορτίο 1, Σκουπίδια 5
Βήμα 15: Θέση 6, Φορτίο 1, Σκουπίδια 5
Βήμα 16: Θέση 7, Φορτίο 3, Σκουπίδια 3
Βήμα 17: Θέση 6, Φορτίο 3, Σκουπίδια 3
Βήμα 18: Θέση 5, Φορτίο 3, Σκουπίδια 3
Βήμα 19: Θέση 4, Φορτίο 3, Σκουπίδια 3
Βήμα 20: Θέση 3, Φορτίο 3, Σκουπίδια 3
Βήμα 21: Θέση 2, Φορτίο 3, Σκουπίδια 3
Βήμα 22: Θέση 1, Φορτίο 3, Σκουπίδια 3
Βήμα 23: Θέση 1, Φορτίο 0, Σκουπίδια 3
Βήμα 24: Θέση 2, Φορτίο 0, Σκουπίδια 3
Βήμα 25: Θέση 3, Φορτίο 0, Σκουπίδια 3
Βήμα 26: Θέση 4, Φορτίο 0, Σκουπίδια 3
Βήμα 27: Θέση 5, Φορτίο 0, Σκουπίδια 3
Βήμα 28: Θέση 6, Φορτίο 0, Σκουπίδια 3
Βήμα 29: Θέση 7, Φορτίο 0, Σκουπίδια 3
Βήμα 30: Θέση 8, Φορτίο 3, Σκουπίδια 0
Βήμα 31: Θέση 7, Φορτίο 3, Σκουπίδια 0
Βήμα 32: Θέση 6, Φορτίο 3, Σκουπίδια 0
Βήμα 33: Θέση 5, Φορτίο 3, Σκουπίδια 0
Βήμα 34: Θέση 4, Φορτίο 3, Σκουπίδια 0
Βήμα 35: Θέση 3, Φορτίο 3, Σκουπίδια 0
Βήμα 36: Θέση 2, Φορτίο 3, Σκουπίδια 0
Βήμα 37: Θέση 1, Φορτίο 3, Σκουπίδια 0
Βήμα 38: Θέση 1, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 123
Μέθοδος: BFS
```

Ευριστική:

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

Μονοπάτι λύσης (47 βήματα):
Βήμα 0: Θέση 1, Φορτίο 0, Σκουπίδια 10
Βήμα 1: Θέση 2, Φορτίο 3, Σκουπίδια 7
Βήμα 2: Θέση 1, Φορτίο 3, Σκουπίδια 7
Βήμα 3: Θέση 1, Φορτίο 0, Σκουπίδια 7
Βήμα 4: Θέση 2, Φορτίο 0, Σκουπίδια 7
Βήμα 5: Θέση 3, Φορτίο 0, Σκουπίδια 7
Βήμα 6: Θέση 4, Φορτίο 1, Σκουπίδια 6
Βήμα 7: Θέση 3, Φορτίο 1, Σκουπίδια 6
Βήμα 8: Θέση 2, Φορτίο 1, Σκουπίδια 6
Βήμα 9: Θέση 1, Φορτίο 1, Σκουπίδια 6
Βήμα 10: Θέση 1, Φορτίο 0, Σκουπίδια 6
Βήμα 11: Θέση 2, Φορτίο 0, Σκουπίδια 6
Βήμα 12: Θέση 3, Φορτίο 0, Σκουπίδια 6
Βήμα 13: Θέση 4, Φορτίο 0, Σκουπίδια 6
Βήμα 14: Θέση 5, Φορτίο 1, Σκουπίδια 5
Βήμα 15: Θέση 4, Φορτίο 1, Σκουπίδια 5
Βήμα 16: Θέση 3, Φορτίο 1, Σκουπίδια 5
Βήμα 17: Θέση 2, Φορτίο 1, Σκουπίδια 5
Βήμα 18: Θέση 1, Φορτίο 1, Σκουπίδια 5
Βήμα 19: Θέση 1, Φορτίο 0, Σκουπίδια 5
Βήμα 20: Θέση 2, Φορτίο 0, Σκουπίδια 5
Βήμα 21: Θέση 3, Φορτίο 0, Σκουπίδια 5
Βήμα 22: Θέση 4, Φορτίο 0, Σκουπίδια 5
Βήμα 23: Θέση 5, Φορτίο 0, Σκουπίδια 5
Βήμα 24: Θέση 6, Φορτίο 0, Σκουπίδια 5
Βήμα 25: Θέση 7, Φορτίο 2, Σκουπίδια 3
Βήμα 26: Θέση 6, Φορτίο 2, Σκουπίδια 3
Βήμα 27: Θέση 5, Φορτίο 2, Σκουπίδια 3
Βήμα 28: Θέση 4, Φορτίο 2, Σκουπίδια 3
Βήμα 29: Θέση 3, Φορτίο 2, Σκουπίδια 3
Βήμα 30: Θέση 2, Φορτίο 2, Σκουπίδια 3
Βήμα 31: Θέση 1, Φορτίο 2, Σκουπίδια 3
Βήμα 32: Θέση 1, Φορτίο 0, Σκουπίδια 3
Βήμα 33: Θέση 2, Φορτίο 0, Σκουπίδια 3
Βήμα 34: Θέση 3, Φορτίο 0, Σκουπίδια 3
Βήμα 35: Θέση 4, Φορτίο 0, Σκουπίδια 3
Βήμα 36: Θέση 5, Φορτίο 0, Σκουπίδια 3
Βήμα 37: Θέση 6, Φορτίο 0, Σκουπίδια 3
Βήμα 38: Θέση 7, Φορτίο 0, Σκουπίδια 3
Βήμα 39: Θέση 8, Φορτίο 3, Σκουπίδια 0
Βήμα 40: Θέση 7, Φορτίο 3, Σκουπίδια 0
Βήμα 41: Θέση 6, Φορτίο 3, Σκουπίδια 0
Βήμα 42: Θέση 5, Φορτίο 3, Σκουπίδια 0
Βήμα 43: Θέση 4, Φορτίο 3, Σκουπίδια 0
Βήμα 44: Θέση 3, Φορτίο 3, Σκουπίδια 0
Βήμα 45: Θέση 2, Φορτίο 3, Σκουπίδια 0
Βήμα 46: Θέση 1, Φορτίο 3, Σκουπίδια 0
Βήμα 47: Θέση 1, Φορτίο 0, Σκουπίδια 0

Συνολικές καταστάσεις που εξετάστηκαν: 54
Μέθοδος: Ευριστική (Best-First Search)
```

Σε αυτήν την περίπτωση οι τυφλοί αλγόριθμοι μας δίνουν τα αναμενόμενα αποτελέσματα. Ενδιαφέρον όμως παρουσιάζει η Best-First Search, η οποία στο συγκεκριμένο πρόβλημα δεν ήταν τόσο αποδοτική όσο θα περιμέναμε.

Η DFS βρίσκει την λύση χωρίς να εξετάσει πολλές καταστάσεις, απέχει όμως από την βέλτιστη.

Η BFS βρίσκει την βέλτιστη λύση με διαφορά 8 βημάτων αλλά εξετάζει πάνω από τις διπλάσιες καταστάσεις που εξέτασε η DFS.

Τέλος, η ευριστική μέθοδος βρίσκει την λύση με μεγαλύτερο αριθμό βημάτων κατά ένα και εξετάζοντας μία επιπλέον κατάσταση, σε σύγκριση με την DFS. Φυσικά απέχει πολύ από την βέλτιστη λύση της BFS.

Σύνοψη

Έχοντας εξετάσει τις αρχικές καταστάσεις:

1. **[3, 2, 3, 0, 0, 2, 0, 1, 8, 3, 0]** (σελ. 9, 10, 15)
2. **[4, 1, 0, 2, 0, 3, 3, 0, 1, 4, 0]** (σελ. 25)
3. **[1, 0, 3, 0, 1, 1, 0, 2, 3, 1, 0]** (σελ. 27)

έχουμε πετύχει μια ολοκληρωμένη παρουσίαση της συμπεριφοράς των τριών μεθόδων αναζήτησης DFS, BFS και Best-First Search. Και στις τρεις καταστάσεις οι τυφλές μέθοδοι αναζήτησης λειτούργησαν όπως περιμέναμε, δηλαδή:

- η DFS να εξετάζει πολύ λιγότερες καταστάσεις αλλά να μην βρίσκει την βέλτιστη διαδρομή
- η BFS να καθυστερεί αισθητά στην αναζήτηση αλλά να βρίσκει την βέλτιστη διαδρομή

Εκεί που υπήρξε ενδιαφέρον ήταν στα διαφορετικά αποτελέσματα της Best-First μεθόδου σε κάθε κατάσταση. Συγκεκριμένα:

Στην πρώτη κατάσταση, βρήκε την ίδια βέλτιστη διαδρομή που βρήκε και η BFS εξετάζοντας όμως μόλις 42 καταστάσεις αντί για 308.

Στην δεύτερη κατάσταση, δεν κατάφερε να βρει την βέλτιστη λύση καταλήγοντας στο ίδιο μονοπάτι με την DFS, εξετάζοντας όμως τρία λιγότερα βήματα από αυτήν.

Στην τρίτη κατάσταση, η Best-First και κατέληξε σε μεγαλύτερο μονοπάτι αλλά και εξέτασε περισσότερες καταστάσεις συγκριτικά με την DFS. Μπορεί οι διαφορές και στα δύο να ήταν μόνο ένα βήμα και μία κατάσταση παραπάνω αντίστοιχα, αλλά ακόμα κι έτσι η ευριστική μέθοδος απέτυχε να καθοδηγήσει την αναζήτηση πιο «έξυπνα», δίνοντας μας χειρότερα αποτελέσματα από την «τυφλή» DFS.

Συμπερασματικά, στο πρόβλημα μας η ευριστική αναζήτηση δεν είναι η χρυσή τομή γρήγορης αναζήτησης και βέλτιστης διαδρομής που θα θέλαμε. Συγκεκριμένα για την Best-First Search που υλοποιήσαμε, το να γνωρίζει μόνο το πόσο απέχει από την λύση (υπολειπόμενα σκουπίδια και απόσταση από την βάση) δεν αρκεί για να έχουμε σταθερά καλύτερα αποτελέσματα από τις τυφλές μεθόδους, πόσο μάλλον να βρίσκουμε πάντα την βέλτιστη λύση.