Απαλλακτική Εργασία Τεχνητή Νοημοσύνη

Πανεπιστήμιο Δυτικής Αττικής

Σχολή Μηχανικών

Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

5ο εξάμηνο
Ακ. Έτος: 2024-25

Απαλλακτική Εργασία Εξαμήνου Μέρος Ι

Στοιχεία Φοιτητή:
Ονοματεπώνυμο: Λιν Χονγκ Τσε
ΑΜ: 21390120

Περιεχόμενα

1. Να οριστούν και να αναπαρασταθούν κατάλληλα:

i. Ο Κόσμος του Προβλήματος

Ο κόσμος του προβλήματος αποτελείται από μια σειρά 6 διαδοχικών κελιών. Κάθε κελί μπορεί να περιέχει:

- Ένα φαγώσιμο φρούτο (f)
- Ένα δηλητηριώδες φρούτο (d)
- Τίποτα (κενό κελί)
- Τον Pacman (p)

Ο Pacman μπορεί να κινείται σε διπλανό κελί (αριστερά ή δεξιά), να τρώει φαγώσιμα φρούτα ή να καταστρέφει δηλητηριώδη φρούτα. Όταν καταστρέφει το δηλητηριώδες φρούτο, δημιουργείται ένα νέο φαγώσιμο φρούτο σε ένα τυχαίο κενό κελί. Υπάρχει μόνο 1 Pacman , 1 δηλητηριώδες φρούτο (αφού δεν υπάρχει τρόπος να δημιουργηθεί ή να καταστραφεί) και μέχρι 3 φαγώσιμα φρούτα (αφού η αρχική κατάσταση έχει 2 ήδη φρούτα και μπορεί να δημιουργηθεί 1 φαγώσιμο φρούτο καταστρέφοντας το δηλητηριώδες φρούτο, χωρίς να φαγωθεί άλλο φρούτο) ανά πάσα στιγμή.

Αντικείμενα	**Ιδιότητες**	**Σχέσεις**
Pacman	πιάνει ένα κελί,κινείται σε διπλανά κελιά	τρώει φαγώσιμα φρούτα,καταστρέφει δηλητηριώδη φρούτα
φαγώσιμο φρούτο	πιάνει ένα κελί	τρώγεται από Pacman
δηλητηριώδες φρούτο	πιάνει ένα κελί	καταστρέφεται από Pacman και δημιουργεί ένα φαγώσιμο φρούτο σε ένα τυχαίο άδειο κελί
Τίποτα	κενό κελί	

ii. Η αρχική κατάσταση και η τελική κατάσταση του προβλήματος

Αρχική Κατάσταση:

Η πίστα στην αρχή είναι:

1. Δηλητηριώδες φρούτο (d)
2. Φαγώσιμο φρούτο (f)
3. Pacman (p)
4. Κενό ( )
5. Φαγώσιμο φρούτο (f)
6. Κενό ( )

Αυτό σημαίνει ότι ο Pacman ξεκινά στο τρίτο κελί, το δηλητηριώδες φρούτο είναι στο πρώτο κελί, και τα φαγώσιμα φρούτα είναι στα κελιά 2 και 5:

[d, f , p , , f,]

Τελική Κατάσταση:

Ο στόχος είναι να φαγωθούν όλα τα φρούτα και να καταστραφεί το δηλητηριώδες φρούτο, αφήνοντας τον Pacman μόνο του στην πίστα.

Η τελική κατάσταση είναι οποιαδήποτε μορφή της πίστας όπου ο Pacman βρίσκεται μόνος του, δηλαδή μία από τις επόμενες:

[p, , , , ,] [,p, , , ,] [, ,p, , ,] [, , ,p, ,] [, , , ,p,] [, , , , ,p]

iii. Ο χώρος καταστάσεων του προβλήματος (περιγραφικά και με λίστες), παραθέτοντας αντιπροσωπευτικά παραδείγματα.

Περιορισμοί:

Κάθε κελί μπορεί να έχει τον Pacman και ένα μόνο από τα 2 πιθανά περιεχόμενα: φαγώσιμο φρούτο ή δηλητηριώδες φρούτο.

Ο χώρος των καταστάσεων αποτελείται από όλες τις δυνατές διατάξεις των 6 κελιών,όπου το κάθε κελί μπορεί να έχει μόνο ένα αντικείμενο μαζί ή χωρίς τον Pacman. Έτσι ο Pacman μπορεί να είναι σε κελί μαζί με ένα από τα φρούτα αλλά δεν γίνεται να υπάρχουν δύο φρούτα σε ένα κελί.

Επιπλέον μπορεί να υπάρχει μόνο 1 Pacman ανά στιγμή, 1 δηλητηριώδες φρούτο (αφού δεν υπάρχει τρόπος να δημιουργηθεί ή να καταστραφεί) και μέχρι 3 φαγώσιμα φρούτα (μπορεί να δημιουργηθεί 1 φαγώσιμο φρούτο καταστρέφοντας το δηλητηριώδες φρούτο, χωρίς να φαγωθεί άλλο φρούτο).

Αντιπροσωπευτικές Καταστάσεις:

1. Αρχική Κατάσταση:

   ```
   [ d, f, p,  , f,  ]
   ```

2. Μια Ενδιάμεση Κατάσταση μετά από κάποιες κινήσεις:

   ```
   [  ,  , p,  , f, f ]
   ```

   (Ο Pacman έχει φάει το φαγώσιμο φρούτο στο κελί 2, έχει καταστρέψει το φρούτο στο κελί 1 και ως αποτέλεσμα δημιουργήθηκε τυχαία φαγώσιμο φρούτο στο κελί 6. Έπειτα ο Pacman μετακινήθηκε στο κελί 2 και έπειτα στο κελί 3.)

3. Άλλη Μια Ενδιάμεση Κατάσταση μετά από κάποιες κινήσεις:

   ```
   [ d, f+p,  ,  ,  ,  ]
   ```

(Ο Pacman έχει φάει το φαγώσιμο φρούτο στο κελί 5 και μετακινήθηκε στο κελί 2.)

4. Τελική Κατάσταση:

```
[  ,  ,  ,  ,  , p ]
```

Περιγραφή του Χώρου Καταστάσεων με Λίστες:

Ο χώρος των καταστάσεων αποτελείται από την πίστα μας που θα συμβολίζεται ως μια λίστα από 6 κελιά. Το κάθε κελί θα είναι μια λίστα από 2 αντικείμενα. **Το πρώτο στοιχείο** αναπαριστά τον Pacman (με την τιμή p αν είναι παρών στο κελί ή κενό αν δεν είναι). **Το δεύτερο στοιχείο** αναπαριστά το φρούτο (με την τιμή f για το φαγώσιμο φρούτο, d για το δηλητηριώδες ή κενό αν το κελί δεν περιέχει φρούτο).

Ο πίνακας [[X1,Y1] , [X2,Y2] , [X3,Y3] , [X4,Y4] , [X5,Y5] , [X6,Y6]] αναπαριστά την πίστα.

Οι τιμές Xi μπορούν να είναι:

- p: Ο Pacman υπάρχει(μόνο μία φορά)
- : Ο Pacman δεν υπάρχει

Οι τιμές Yi μπορούν να είναι:

- f: Φαγώσιμο φρούτο(μέχρι 3)
- d: Δηλητηριώδες φρούτο(μέχρι 1)
- : Δεν υπάρχει φρούτο

Ο χώρος καταστάσεων περιλαμβάνει όλες τις πιθανές συνδυαστικές τιμές για κάθε κελί, λαμβάνοντας υπόψη τους παραπάνω περιορισμούς.

2. Να περιγραφούν οι τελεστές μετάβασης. Να παρουσιαστεί σχολιασμένη η κωδικοποίηση των τελεστών μετάβασης του προβλήματος και η συνάρτηση εύρεσης απογόνων (findchildren):

Τελεστές Μετάβασης:

Οι τελεστές που περιγράφουν τις ενέργειες του Pacman είναι:

T1: Μετακίνηση αριστερά (move_left): Ο Pacman κινείται στο κελί στα αριστερά του.

Προϋποθέσεις: Να υπάρχει κελί στα αριστερά του Pacman.

Μοντέλο Μετάβασης: Ο Pacman βρίσκεται πλέον στο κελί στα αριστερά όπου βρισκόταν πριν.

T2: Μετακίνηση δεξιά (move_right): Ο Pacman μπορεί να κινηθεί στο κελί στα δεξιά του.

Προϋποθέσεις: Να υπάρχει κελί στα δεξιά του Pacman.

Μοντέλο Μετάβασης: Ο Pacman βρίσκεται πλέον στο κελί στα δεξιά όπου βρισκόταν πριν.

T3: Φάγωμα φρούτου (eat_fruit): Ο Pacman τρώει το φαγώσιμο φρούτο.

Προϋποθέσεις: Να υπάρχει φαγώσιμο φρούτο στο κελί όπου βρίσκεται ο Pacman.

Μοντέλο Μετάβασης: Το φαγώσιμο φρούτο στο κελί του Pacman δεν υπάρχει πια.

T4: Καταστροφή δηλητηριώδους φρούτου (destroy_poison): Ο Pacman καταστρέφει το δηλητηριώδες φρούτο.

Προϋποθέσεις: Να υπάρχει δηλητηριώδες φρούτο στο κελί όπου βρίσκεται ο Pacman.

Μοντέλο Μετάβασης: Το δηλητηριώδες φρούτο στο κελί του Pacman δεν υπάρχει πια. Δημιουργείται ένα νέο φαγώσιμο φρούτο σε τυχαίο κενό κελί.

Ορισμός Κατάστασης & Τρόπος Κωδικοποίησης:

Κάθε κατάσταση στο πρόβλημα ορίζεται από μια λίστα που περιέχει 6 λίστες, μία για κάθε κελί του προβλήματος. Έπειτα κάθε μία από τις 6 λίστες περιέχει δύο τιμές. Η πρώτη τιμή μπορεί να περιέχει f(φαγώσιμο φρούτο), d(δηλητηριώδες φρούτο) ή να είναι κενή. Η δεύτερη τιμή μπορεί να περιέχει μόνο τον Pacman(p) ή να είναι κενή. Και φυσικά όπως αναφέρθηκε παραπάνω υπάρχει μέχρι μόνο 1 Pacman, 1 δηλητηριώδες φρούτο και 3 φαγώσιμα φρούτα ανά κατάσταση.

Έτσι για παράδειγμα η αρχική κατάσταση είναι:

[[' ', 'd'], [' ', 'f'], ['p', ' '], [' ', ' '], [' ', 'f'], [' ', ' ']]

Βοηθητική Συνάρτηση για Εύρεση του Pacman:

```
"""Βοηθητική συνάρτηση που βρίσκει τη θέση του Pacman στην πίστα."""
def find_pacman(state):
    for i in range(len(state)):
        if state[i][0] == 'p':
            return i
    return -1  # Pacman δεν βρέθηκε, κάτι πήγε λάθος
```

Κωδικοποίηση Τελεστών Μετάβασης:

```
"""Μετακινεί τον Pacman ένα κελί αριστερά αν είναι εφικτό."""
def move_left(state):
    pac_pos = find_pacman(state)  # Βρίσκουμε τη θέση του Pacman
    if pac_pos > 0:  # Αν δεν είναι στο πρώτο κελί
        state[pac_pos][0] = ' '
        # Ανταλλάσσουμε τις θέσεις του Pacman
        state[pac_pos-1][0] = 'p'
        return state
    else:
            return None  # Δεν μπορεί να κινηθεί αριστερά

"""Μετακινεί τον Pacman ένα κελί δεξιά αν είναι εφικτό."""
def move_right(state):
    pac_pos = find_pacman(state)  # Βρίσκουμε τη θέση του Pacman
    if pac_pos < len(state)-1:  # Αν δεν είναι στο τελευταίο κελί
        state[pac_pos][0] = ' '
        # Ανταλλάσσουμε τις θέσεις του Pacman
        state[pac_pos+1][0] = 'p'
        return state
    else:
            return None  # Δεν μπορεί να κινηθεί δεξιά

"""Ο Pacman τρώει το φαγώσιμο φρούτο αν βρίσκεται στο ίδιο κελί."""
def eat_fruit(state):
    pac_pos = find_pacman(state)
    if state[pac_pos][1] == 'f':  # Αν υπάρχει φαγώσιμο φρούτο στο κελί
του Pacman
        state[pac_pos][1] = ' '  # Αφαιρούμε το φρούτο
        return state
    else:
            return None  # Δεν υπάρχει φρούτο για να φάει

"""Ο Pacman καταστρέφει το δηλητηριώδες φρούτο και δημιουργεί νέο φαγώσ
ιμο φρούτο."""
def destroy_poison(state):
    pac_pos = find_pacman(state)
    if state[pac_pos][1] == 'd':  # Αν υπάρχει δηλητηριώδες φρούτο στο
κελί του Pacman
        state[pac_pos][1] = ' '  # Καταστρέφουμε το δηλητηριώδες φρούτο

        # Βρίσκουμε τυχαίο κενό κελί χωρίς τον Pacman για να τοποθετήσο
υμε το νέο φαγώσιμο φρούτο
        empty_cells = [i for i in range(len(state)) if state[i][0] !=
'p' and state[i][1] == ' ']
        if empty_cells:
```

```
            new_fruit_pos = random.choice(empty_cells)  # Τυχαία επιλογ
ή κενού κελιού
            # Κάναμε import random στην αρχή
            state[new_fruit_pos][1] = 'f'  # Τοποθετούμε το νέο φρούτο
        return state
    else:
        return None  # Δεν υπάρχει δηλητηριώδες φρούτο να καταστραφεί
```

Συνάρτηση Εύρεσης Απογόνων (findchildren)

```
"""Επιστρέφει όλες τις έγκυρες επόμενες καταστάσεις (απογόνους) από την
τρέχουσα κατάσταση."""
def find_children(state):
    children = []

    # Δοκιμάζουμε την κίνηση αριστερά
    left_state=copy.deepcopy(state)
    child_left = move_left(left_state)
    if not child_left==None:
        children.append(child_left)

    # Δοκιμάζουμε την κίνηση δεξιά
    right_state=copy.deepcopy(state)
    child_right = move_right(right_state)
    if not child_right==None:
        children.append(child_right)

    # Δοκιμάζουμε το φάγωμα φρούτου
    eat_state=copy.deepcopy(state)
    child_eat = eat_fruit(eat_state)
    if not child_eat==None:
        children.append(child_eat)

    # Δοκιμάζουμε την καταστροφή δηλητηριώδους φρούτου
    destroy_state=copy.deepcopy(state)
    child_destroy = destroy_poison(destroy_state)
    if not child_destroy==None:
        children.append(child_destroy)

    return children  # Επιστρέφουμε τη λίστα με όλες τις έγκυρες καταστ
άσεις
```

3.Επιλύστε το παιχνίδι Pacman όπως περιγράφεται παραπάνω, χρησιμοποιώντας τον αλγόριθμο της πρώτα σε βάθος αναζήτησης (DFS) και της πρώτα σε πλάτος αναζήτησης (BFS) με παρακολούθηση μετώπου για εύρεση του στόχου, παρουσιάστε και σχολιάστε τα αποτελέσματα:

Θα χρειαστούμε πρώτα μια συνάρτηση που ελέγχει αν η τρέχουσα κατάσταση είναι η τελική κατάσταση:

```
"""Ελέγχει αν η τρέχουσα κατάσταση είναι η τελική κατάσταση (στόχο
ς)."""
def is_goal_state(state):
    # Μετράει πόσα κελιά περιέχουν Pacman
    pacman_count = 0

    for cell in state: # Για κάθε κελί(λίστα με δύο στοιχεία) μέσα σε κ
άθε κατάσταση(λίστα με 6 λίστες)
        pacman, fruit = cell # όπου το 1ο στοιχείο κάθε κελιού συμβολίζ
εται με την μεταβλητή pacman και το 2ο στοιχείο κάθε κελιού με την μετα
βλητή fruit
        if fruit in ['f', 'd']: # Ελέγχει αν υπάρχουν φρούτα(φαγώσιμα ή
δηλητηριώδες)
            return False
        # Μετράει πόσες φορές εμφανίζεται ο Pacman
        if pacman == 'p':
            pacman_count += 1

    # Ο Pacman θα πρέπει να εμφανίζεται ακριβώς μία φορά
    return pacman_count == 1
```

Για να είναι πιο ορατά τα αποτελέσματα χρησιμοποιούμε βοηθητική συνάρτηση αναπαράστασης των αποτελεσμάτων:

```
""" ----------------------------------------------------------------
----------
** Αναπαράσταση Αποτελεσμάτων
"""
def print_state(state):
```

```
    visual = []
    for cell in state:
        pacman = 'P' if cell[0] == 'p' else '.'  # Αν υπάρχει ο pacman
        fruit = 'F' if cell[1] == 'f' else 'D' if cell[1] == 'd' else
'.'  # Αν υπάρχει φρούτο
        if pacman == 'P' and fruit != '.':
            visual.append(f'{pacman}+{fruit}')  # Pacman με φρούτο
        elif pacman == 'P':
            visual.append(pacman)  # Pacman μόνος του
        elif fruit != '.':
            visual.append(fruit)  # Φρούτο μόνο του
        else:
            visual.append('.')  # Κενό κελ

    print(' | '.join(visual))  # Εμφανίζει την πίστα
```

Αλγόριθμοι πρώτα σε βάθος αναζήτησης (DFS) και πρώτα σε πλάτος αναζήτησης (BFS) με παρακολούθηση μετώπου:

```
"""
** Αρχικοποίηση Μετώπου
"""

def make_front(state):
    return [state]   # Το μέτωπο είναι λίστα με τις καταστάσεις προς αν
άλυση

""" ----------------------------------------------------------------
---------
**** Επέκταση Μετώπου
"""

def expand_front(front, method):
    if method=='DFS':
        if front:
            print("Front states:")
            for state in front:
                print_state(state)
            node=front.pop(0)   # Αφαιρούμε τον πρώτο κόμβο από το μέτω
πο
            for child in find_children(node):   # Εύρεση των παιδιών (α
πογόνων) της κατάστασης
                front.insert(0,child)   # Προσθέτουμε τα παιδιά στην AP
ΧΗ του μετώπου
```

```
    elif method=='BFS':
         if front:
            print("Front states:")
            for state in front:
                print_state(state)
            node=front.pop(0)
            for child in find_children(node):
                front.append(child)     # Προσθέτουμε τα παιδιά στο ΤΕΛ
ΟΣ του μετώπου
    #elif method=='BestFS':

    return front


""" -----------------------------------------------------------------
---------
**** Βασική αναδρομική συνάρτηση για δημιουργία δέντρου αναζήτησης (ανα
δρομική επέκταση δέντρου)
"""
def find_solution(front, closed, method, step=0):
    if not front:  # Αν το μέτωπο είναι κενό, δεν βρέθηκε λύση
        print('No solution')

    elif front[0] in closed:  # Αν η πρώτη κατάσταση έχει ήδη εξεταστεί
        print("Step:", step)
        print("State Removed:")
        print_state(front[0])
        new_front = copy.deepcopy(front)
        new_front.pop(0)  # Αφαιρούμε την τρέχουσα κατάσταση και συνεχί
ζουμε
        find_solution(new_front, closed, method, step+1)

    elif is_goal_state(front[0]):  # Αν η τρέχουσα κατάσταση είναι ο στ
όχος
        print('Goal found in', step, 'steps!')
        print("Final State:")
        print_state(front[0])  # Εμφανίζει την τελική κατάσταση της πίσ
τας

    else:
        print("Step:", step)
        closed.append(front[0])  # Προσθέτουμε την κατάσταση στη λίστα
των εξετασμένων
        front_copy = copy.deepcopy(front)
        front_children = expand_front(front_copy, method)
```

```
        print("Current State:")
        print_state(front[0])  # Εμφανίζει την τρέχουσα κατάσταση της π
ίστας
        closed_copy = copy.deepcopy(closed)
        find_solution(front_children, closed_copy, method, step+1)
```

Εκτέλεση Κώδικα:

```
""" ----------------------------------------------------------------
----------
** Κλήση Εκτέλεσης Κώδικα
"""

def main():


    initial_state=[[' ','d'],[' ','f'],['p',' '],[' ',' '],[' ','f'],['
',' ']]

    """ ----------------------------------------------------------------
-------------
    **** Επιλογή Μεθόδου Αναζήτησης
    """

    method='DFS'

    """ ----------------------------------------------------------------
-------------
    **** Έναρξη Αναζήτησης
    """

    print('Begin Searching:')
    find_solution(make_front(initial_state), [], method)



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

Παρουσίαση DFS:

Ο αλγόριθμος DFS εξερευνά κάθε κλάδο του δέντρου αναζήτησης όσο βαθύτερα μπορεί, μέχρι να βρει τον στόχο ή να εξαντλήσει τις δυνατότητες του. Μπορεί να βρει λύση γρήγορα

αν η λύση βρίσκεται βαθιά στο δέντρο, όμως δεν εγγυάται πως θα βρει την βέλτιστη λύση. Όμως χρησιμοποιεί λιγότερη μνήμη από το BFS, αφού αποθηκεύει μόνο τη διαδρομή του τρέχοντος κλάδου που εξετάζει.

Στον κώδικα στην συνάρτηση find_children η σειρά των τελεστών μετάβασης είναι move_left → move_right → eat_fruit → destroy_fruit. Επειδή στην συνάρτηση expand_front για την μέθοδο DFS τα παιδιά κόμβοι τοποθετούνται πάντα στην αρχή, η σειρά που θα εμφανίζονται οι καταστάσεις στο μέτωπο θα είναι οι απόγονοι της Current State σε ανάποδη σειρά, δηλαδή destroy_fruit → eat_fruit → move_right → move_left αν μπορούν να εκτελεστούν φυσικά. Μετά ακολουθούν οι καταστάσεις που έμειναν από πριν στους άλλους κλάδους.

Τρέχοντας το πρόγραμμα παίρνουμε:

```
Begin Searching:
Step: 0
Front states:
D | F | P | . | F | . #Η Αρχική Κατάσταση
Current State:
D | F | P | . | F | . #Είναι και η τρέχων κατάσταση που θα εφαρμόσουμε
τους τελεστές μετάβασης για να παράγουμε τις καταστάσεις παιδιά της και
μετά θα την αφεραίσουμε από το μέτωπο
Step: 1
Front states:        #Από την προϊγούμενη κατάσταση
D | F | . | P | F | . #Pacman move right
D | P+F | . | . | F | . #Pacman move left
Current State:
D | F | . | P | F | . #Θα πάρουμε αυτή και θα εφαρμόσουμε τους τελεστές
μετάβασης και έπειτα θα την αφαιρέσουμε από το μέτωπο. Πάντα παίρνουμε
την 1η κατάσταση με τον αλγόριθμο DFS
Step: 2
Front states:
D | F | . | . | P+F | . #Pacman move right
D | F | P | . | F | . #Pacman move left. Παρατηρούμε επίσης πως είναι ί
δια με την αρχική μας κατάσταση που σημαίνει ότι είναι ήδη στο κελιστό
σύνολο. Άρα όταν (και αν) γίνει αυτή current state απλά θα την αφαιρέσο
υμε από το μέτωπο
D | P+F | . | . | F | . #Έμεινε απο πρίν
Current State:
D | F | . | . | P+F | .
Step: 3
Front states:
D | F | . | . | P | . #Επειδή υπάρχει φρούτο στο ίδιο κελί με τον Pacma
n, τώρα μπορεί να γίνει Pacman eat fruit
D | F | . | . | F | P #Pacman move right
D | F | . | P | F | . #Pacman move left
D | F | P | . | F | . #Έμειναν απο πρίν
D | P+F | . | . | F | .
```

```
Current State:
D | F | . | . | P | .
Step: 4
Front states:
D | F | . | . | . | P #Pacman move right
D | F | . | P | . | . #Pacman move left
D | F | . | . | F | P #Έμειναν απο πρίν
D | F | . | P | F | .
D | F | P | . | F | .
D | P+F | . | . | F | .
Current State:
D | F | . | . | . | P #Η μόνη ενέργεια που μπορεί να κάνει εδώ ο Pacman
είναι move left
Step: 5
State Removed:
D | F | . | . | P | . #Η κατάσταση αυτή βρέθηκε ήδη στο βήμα 3 άρα αφαι
ρείται απλά
Step: 6
Front states:
D | F | . | P | . | . #Έμειναν απο πρίν
D | F | . | . | F | P
D | F | . | P | F | .
D | F | P | . | F | .
D | P+F | . | . | F | .
Current State:
D | F | . | P | . | .
Step: 7
State Removed:
D | F | . | . | P | . #Η move right αφαιρείται
Step: 8
Front states:
D | F | P | . | . | . #Pacman move left
D | F | . | . | F | P #Έμειναν απο πρίν
D | F | . | P | F | .
D | F | P | . | F | .
D | P+F | . | . | F | .
Current State:
D | F | P | . | . | .
Step: 9
State Removed:
D | F | . | P | . | . #Η move right αφαιρείται
Step: 10
Front states:
D | P+F | . | . | . | . #Pacman move left
D | F | . | . | F | P #Έμειναν απο πρίν κ.ο.κ. Με την ίδια λογική συνεχ
ίζουμε μέχρι να βρούμε μια τελική κατάσταση. Το υπόλοιπο μέρος του αποτ
```

```
ελέσματος δίνεται στο άλλο pdf
D | F | . | P | F | .
D | F | P | . | F | .
D | P+F | . | . | F | .


Current State:
. | . | . | P+F | . | .
Goal found in 25 steps!
Final State:
. | . | . | P | . | . #Βρέθηκε Τελική Κατάσταση
```

Σχολιασμός Αποτελεσμάτων DFS:

Στο πρόβλημα του Pacman δεν υπάρχουν ενέργειες(τελεστές μετάβασης) που να κάνουν απίθανη την ολοκλήρωση του παιχνιδιού. Έτσι με την χρήση του αλγορίθμου DFS χρειάζεται να εξερευνήσουμε **έναν** κλάδο του δέντρου μόνο κάθε φορά, πράγμα που κάνει τον αλγόριθμο DFS πιο αποδοτικό από τον αλγόριθμο BFS στη συγκεκριμένη περίπτωση.

Παρουσίαση BFS:

Αλλάζουμε στο πρόγραμμα την μεταβλητή method='BFS'

Ο αλγόριθμος BFS εξερευνά τα επίπεδα του δέντρου αναζήτησης ένα προς ένα, εξασφαλίζοντας ότι θα βρει την συντομότερη λύση, αν υπάρχει. Όμως χρησιμοποιεί πολύ περισσότερη μνήμη από το DFS, καθώς αποθηκεύει όλα τα κλαδιά του επιπέδου που εξετάζει και μπορεί να είναι αργό αν η λύση βρίσκεται πολύ βαθιά.

Στον κώδικα στην συνάρτηση find_children η σειρά των τελεστών μετάβασης είναι move_left → move_right → eat_fruit → destroy_fruit. Επειδή στην συνάρτηση expand_front για την μέθοδο BFS τα παιδιά κόμβοι τοποθετούνται πάντα στο τέλος του μετώπου, η σειρά που θα εμφανίζονται οι καταστάσεις στο μέτωπο είναι πρώτα οι καταστάσεις που έμειναν από πριν και έπειτα οι απόγονοι της Current State με σειρά move_left → move_right → eat_fruit → destroy_fruit.

Τρέχοντας το πρόγραμμα παίρνουμε:

```
Begin Searching:
Step: 0
Front states:
D | F | P | . | F | . #Η Αρχική Κατάσταση
Current State:
D | F | P | . | F | .
Step: 1
Front states:
D | P+F | . | . | F | . #Pacman move left
D | F | . | P | F | . #Pacman move right
Current State:
```

```
D | P+F | . | . | F | .
Step: 2
Front states:
D | F | . | P | F | . #Η κατάσταση που έμεινε από πριν είναι τώρα 1η στ
ο μέτωπο. Με την BFS αναλύουμε όλους τους κλάδους του κάθε επιπέδου
P+D | F | . | . | F | . #Pacman move left για την προϊγούμενη τρέχουσα
κατάσταση
D | F | P | . | F | . #Pacman move right
D | P | . | . | F | . #Pacman eat fruit
Current State:
D | F | . | P | F | .
Step: 3
Front states:
P+D | F | . | . | F | . #Οι καταστάσεις που έμειναν από πριν
D | F | P | . | F | . #Είναι ίδια με την αρχική οπότε θα αφαιρεθεί απλά
από το μέτωπο όταν έρθει η ώρα να την αναλύσουμε
D | P | . | . | F | .
D | F | P | . | F | . #Pacman move left για την προϊγούμενη τρέχουσα κα
τάσταση
D | F | . | . | P+F | . #Pacman move right
Current State:
P+D | F | . | . | F | .
Step: 4
State Removed:
D | F | P | . | F | . #Αφαιρείται γιατί είναι ίδια με την αρχική κατάστ
αση και δεν υπάρχει λόγος να πάρουμε τα παιδιά της
Step: 5
Front states:
D | P | . | . | F | . #Οι καταστάσεις που έμειναν από πριν
D | F | P | . | F | .
D | F | . | . | P+F | .
D | P+F | . | . | F | . #Pacman move right
P | F | . | . | F | F #Pacman destroy fruit. Αφού καταστράφηκε το δηλητ
ηριώες φρούτο δημιουργήθηκε τυχαία νέο φαγώσιμο φρούτο στο κελί 6.
Current State:
D | P | . | . | F | .
Step: 6
State Removed:
D | F | P | . | F | . #Αφαιρείται γιατί είναι ίδια με την αρχική κατάστ
αση και δεν υπάρχει λόγος να πάρουμε τα παιδιά της
Step: 7
Front states:
D | F | . | . | P+F | . #Οι καταστάσεις που έμειναν από πριν
D | P+F | . | . | F | .
P | F | . | . | F | F
P+D | . | . | . | F | . #Pacman move left
```

```
D | . | P | . | F | . #Pacman move right
Current State:
D | F | . | . | P+F | .
Step: 8
State Removed:
D | P+F | . | . | F | . #Αφαιρείται γιατί είναι ίδια με την τρέχουσα κα
τάσταση 1
Step: 9
Front states:
P | F | . | . | F | F #Οι καταστάσεις που έμειναν από πριν
P+D | . | . | . | F | .
D | . | P | . | F | .
D | F | . | P | F | . #Pacman move left
D | F | . | . | F | P #Pacman move right
D | F | . | . | P | . #Pacman eat fruit κ.ο.κ. Με παρόμοιο τρόπο συνεχί
ζει ο αλγόριθμος να ψάχνει για τελική κατάσταση. Το υπόλοιπο μέρος του
αποτελέσματος δίνεται στο άλλο pdf

Goal found in 107 steps!
Final State:
. | . | . | . | P | .
```

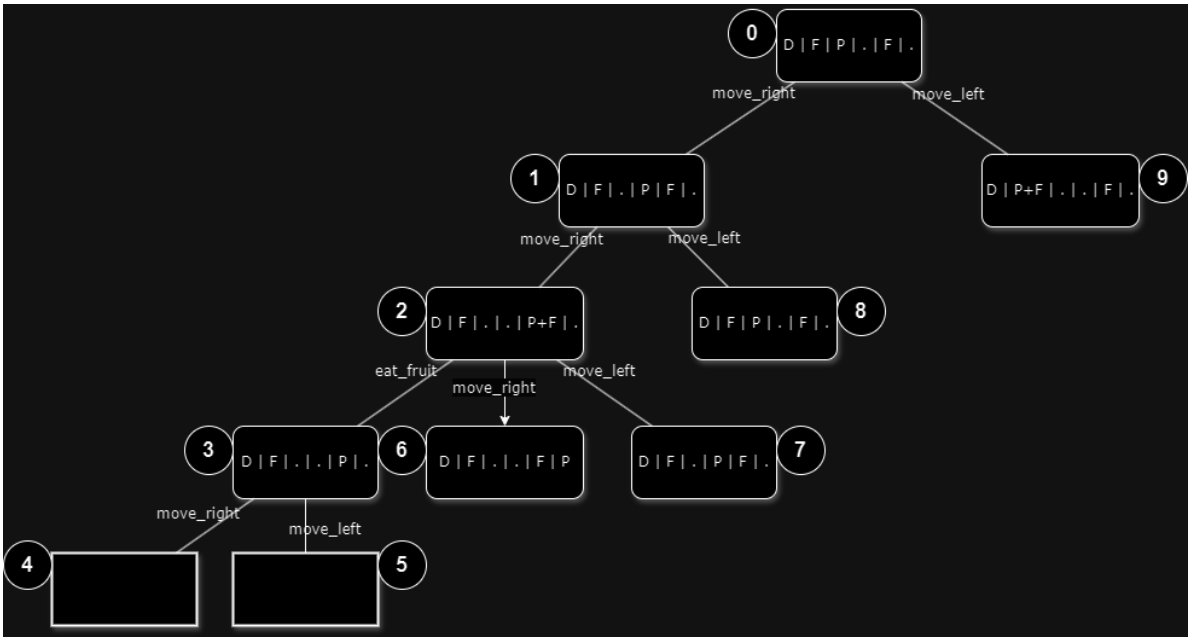
Σχολιασμός Αποτελεσμάτων BFS:

Όπως βλέπουμε το πρόβλημα λύθηκε μέσα σε 107 βήματα με την μέθοδο BFS σε αντίθεση με την μέθοδο DFS που λύθηκε μέσα σε 25 βήματα. Αυτό συμβαίνει γιατί το πρόβλημα χρειάζεται αρκετά βήματα για να καταλήξει σε τελική κατάσταση όπου γίνεται με την μέθοδο DFS, ενώ με την μέθοδο BFS ψάχνουμε ταυτόχρονα πολλά μονοπάτια.

Γι'αυτό τον λόγο μάλιστα βλέπουμε πως προς το τέλος το μέτωπο αναζήτησης είναι γεμάτο καταστάσεις, τρώγοντας έτσι πολύ παραπάνω μνήμη.

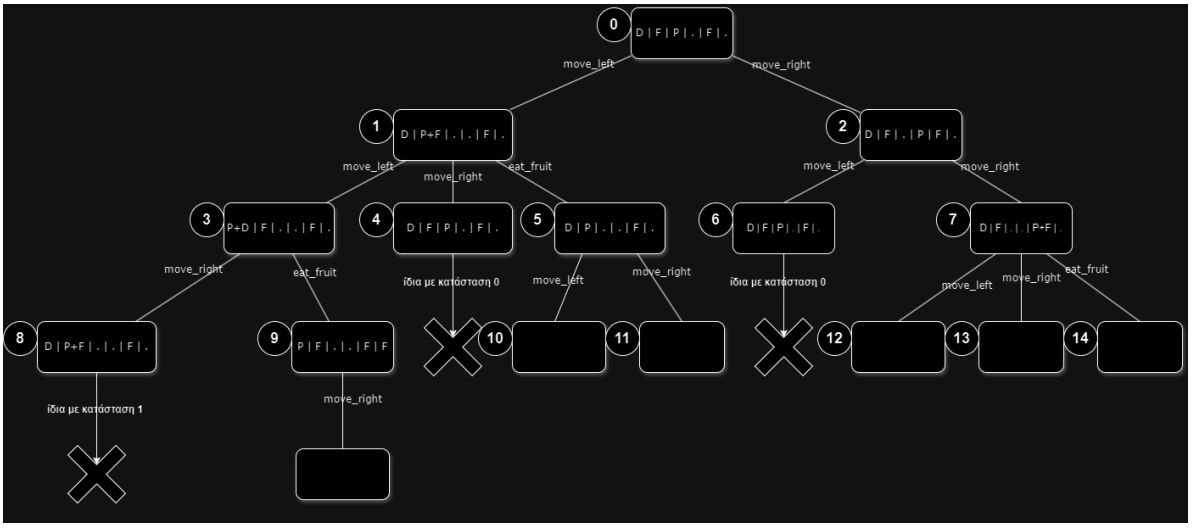
4. Για την DFS και BFS μέθοδο αναζήτησης να σχεδιαστεί το δένδρο αναζήτησης (όχι εξαντλητικό).
Αν το δένδρο προχωρά σε πολύ μεγάλο βάθος περιορίστε το στα πρώτα 4 βήματα ή μέχρι να φαγωθεί ένα φρούτο.

Δένδρο Αναζήτησης DFS:



Τα τετράγωνα αντιπροσωπεύουν μία κατάσταση. Οι αριθμοί δίπλα στα τετράγωνα δείχνουν την σειρά με την οποία η DFS θα επισκεπτόταν τον κάθε κόμβο του δένδρου αν ήταν αυτό το ολικό δένδρο.

Δένδρο Αναζήτησης BFS:



Παρόμοια για την BFS.

5. Εισάγετε κάποιο ευριστικό κριτήριο αναζήτησης, υιοθετώντας μια ευριστική μέθοδο

αναζήτησης για την επίλυση του προβλήματος. Παρουσιάστε και σχολιάστε τα αποτελέσματα.

Ως ευριστικό κριτήριο αναζήτησης θα χρησιμοποιήσουμε ως κύριο κριτήριο την απόσταση του Pacman από το δηλητηριώδες φρούτο καθώς καταστρέφοντας το δηλητηριώδες φρούτο δημιουργείται νέο φρούτο που μπορεί να εμφανιστεί σε οποιοδήποτε άδειο κελί και πιθανότατα σε ένα πολύ απομακρισμένο κελί από τον Pacman. Έτσι καταστρέφοντάς το μπορούμε να σιγουρευτούμε για τις επόμενες κινήσεις μας.

Έπειτα αφού καταστραφεί το δηλητηριώδες φρούτο θα πάρουμε υπόψη την απόσταση του Pacman από τα υπόλοιπα φρούτα που παραμένουν στην πίστα. Όσο μικρότερη η τιμή του ευριστικού κριτηρίου τόσο πιο ψηλά στο μέτωπο θα ανέρχεται η κάθε κατάσταση. Παρόμοια όσο μικρότερη η απόσταση του Pacman από τα φρούτα τόσο πιο κοντά βρισκόμαστε σε τελική κατάσταση.

Αλγόριθμος Best-First-Search:

Προστίθεται στην συνάρτηση def expand_front(front, method) ο ακόλουθος κώδικας για τον αλγόριθμο Best First Search που θα ταξινομεί τους διάδοχους κόμβους στο μέτωπο με βάση το ευριστικό κριτήριό μας:

```
elif method=='BestFS':
        if front:
            print("Front states")
            for state in front:
                print_state(state)
                print(f"Heuristic: {heuristic(state)}\n")   # Τυπώνει τ
ην τιμή του ευριστικού κριτηρίου για κάθε κατάσταση
            node=front.pop(0)
            for child in find_children(node):
                front.append(child)     # Προσθέτουμε τα παιδιά στο μέτ
ωπο
                front.sort(key=lambda state: heuristic(state))
                # Ταξινομούμε το
                # μέτωπο με βάση το ευριστικό κριτήριο
                # (όσο μικρότερο τόσο πιο μπροστά στο μέτωπο τοποθετείτ
αι.
                # Αν μερικές καταστάσεις έχουν την ίδια ευριστική τιμή,
                # διατηρούμε τη σειρά εισαγωγής τους)

    return front
```

Συνάρτηση Ευριστικού Κριτηρίου Αναζήτησης:

```
""" ----------------------------------------------------------------
---------
**** Συνάρτηση Ευριστικού Κριτηρίου Αναζήτησης
"""
def heuristic(state):
    pacman_pos = None   # Θέση του Pacman
    poison_pos = None   # Θέση του δηλητηριώδες φρούτου
    fruit_positions = []    # Θέση όλων των φαγώσιμων φρούτων

    # Βρίσκουμε τις θέσεις όλων των αντικειμένων
    for i in range(len(state)):
        if state[i][0] == 'p':
            pacman_pos = i
        if state[i][1] == 'd':
            poison_pos = i
        if state[i][1] == 'f':
            fruit_positions.append(i)

    # Αν υπάρχει το δηλητηριώδες φρούτο, δώσε προτεραιότητα στο να το κ
αταστρέψεις
    if not poison_pos==None:
        return 13 + abs(pacman_pos - poison_pos) # Θέλουμε να κατστρέψο
υμε το δηλητηριώδες φρούτο πρώτα απ'όλα. Έτσι θα θεωρήσουμε κάθε κατάστ
αση χωρίς το δηλητηριώδες φρούτο ως πιο κοντά στο στόχο μας από οποιαδή
ποτε κατάσταση με το δηλητηριώδες φρούτο. Στην χειρότερη περίπτωση μπορ
ούμε να βρεθούμε στην κατάσταση P| | |F|F|F που επιστρέφει 3+4+5=12. Γ
ι'αυτόν τον λόγο αν υπάρχει δηλητηριώδες φρούτο επιστρέφεται η απόσταση
του Pacman από αυτό + 13.
    else:
    # Αν δεν υπάρχει το δηλητηριώδες φρούτο, μέτρα την συνολική απόστασ
η από όλα τα φρούτα
        total_distance = 0
        for fruit_pos in fruit_positions:
            total_distance += abs(pacman_pos - fruit_pos)
        return total_distance  # Lower distance means closer to the goa
l
```

Παρουσίαση Αποτελεσμάτων:

Αλλάζουμε στην main την μεταβλητή method='BestFS’ και τρέχουμε το πρόγραμμα:

```
Begin Searching:
Step: 0
Front states
```

```
D | F | P | . | F | .
Heuristic: 15 # 15:13+2κελιά απόσταση μεταξύ D και P

Current State:
D | F | P | . | F | .
Step: 1
Front states
D | P+F | . | . | F | .
Heuristic: 14 # Έχει μικρότερο ευριστικό άρα βρίσκεται στην κορυφή του
μετώπο και θα εξεταστεί στο επόμενο βήμα

D | F | . | P | F | .
Heuristic: 16

Current State:
D | P+F | . | . | F | .
Step: 2
Front states
P+D | F | . | . | F | .
Heuristic: 13

D | P | . | . | F | .
Heuristic: 14

D | F | P | . | F | .
Heuristic: 15

D | F | . | P | F | .
Heuristic: 16

Current State:
P+D | F | . | . | F | .
Step: 3
Front states
P | F | F | . | F | .
Heuristic: 7 # Είναι η μόνη κατάσταση χωρίς D άρα αυτόματα έχει το μικρ
ότερο ευριστικό

D | P | . | . | F | .
Heuristic: 14

D | P+F | . | . | F | .
Heuristic: 14

D | F | P | . | F | .
Heuristic: 15
```

```
D | F | . | P | F | .
Heuristic: 16

Current State:
P | F | F | . | F | .
Step: 4
Front states
. | P+F | F | . | F | .
Heuristic: 4

D | P | . | . | F | .
Heuristic: 14

D | P+F | . | . | F | .
Heuristic: 14

D | F | P | . | F | .
Heuristic: 15

D | F | . | P | F | .
Heuristic: 16

Current State:
. | P+F | F | . | F | .
Step: 5
Front states
. | F | P+F | . | F | .
Heuristic: 3 # Εδώ βλέπουμε πως η κίνηση move_right έχει μικρότερο ευρι
στικό από την κίνηση eat_fruit παρά το γεγονός ότι η κίνηση eat_fruit λ
ογικά θα ήταν η αποδοτικότερη(θα φτάναμε σε τελική κατάσταση πιο γρήγορ
α). Όμως με την κίνηση move_right ο Pacman βρίσκεται πιο κοντά στα φρού
τα συνολικά.
Το υπόλοιπο μέρος του αποτελέσματος δίνεται στο άλλο pdf
. | P | F | . | F | .
Heuristic: 4

P | F | F | . | F | .
Heuristic: 7

D | P | . | . | F | .
Heuristic: 14

D | P+F | . | . | F | .
Heuristic: 14
```

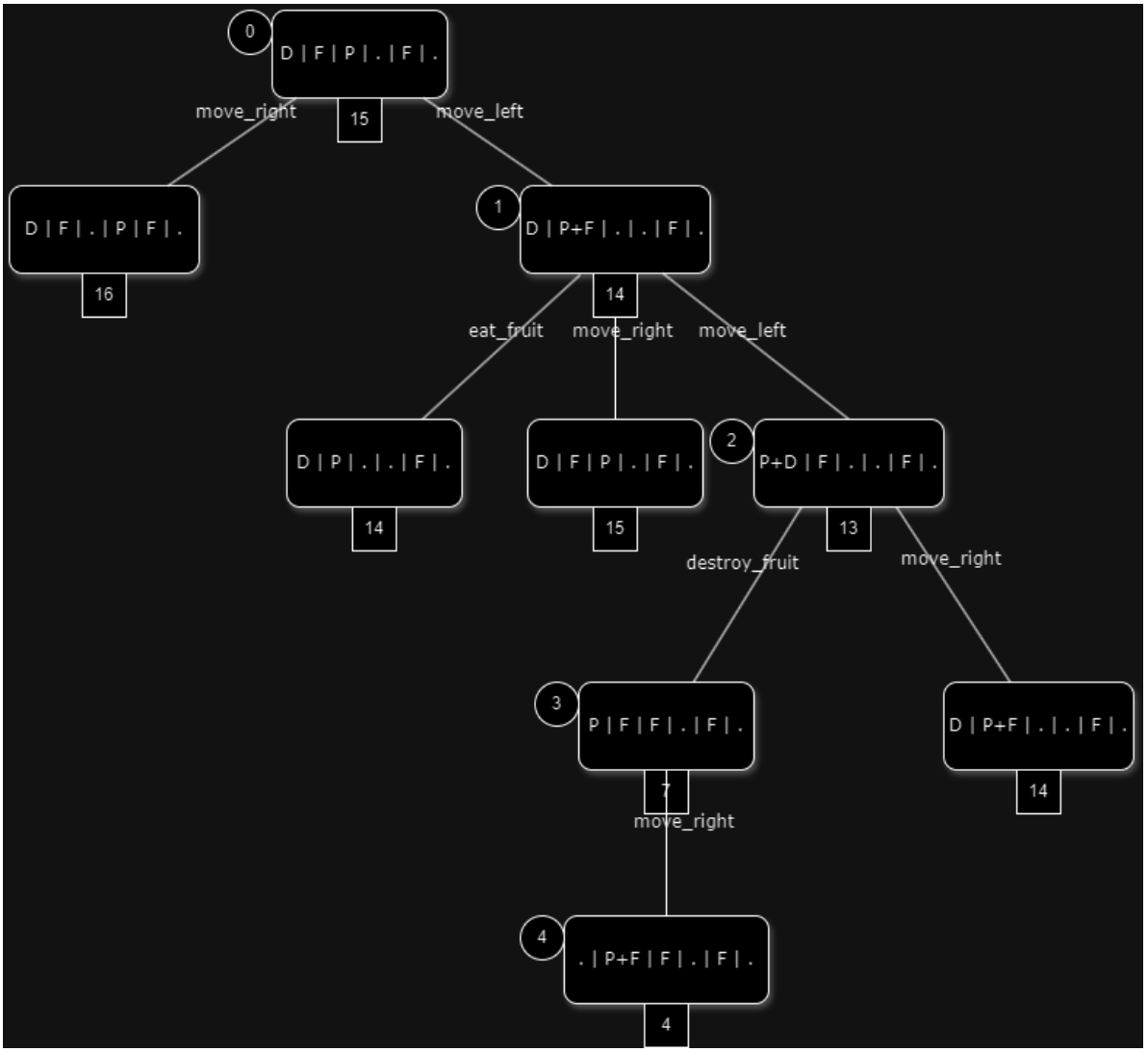
```
D | F | P | . | F | .
Heuristic: 15

D | F | . | P | F | .
Heuristic: 16


Goal found in 14 steps!
Final State:
. | . | . | . | P | .
```

Σχολιασμός Αποτελεσμάτων BestFS:

Γενικά βλέπουμε πως στα πρώτα βήματα οι καταστάσεις με τον Pacman να βρίσκεται πιο κοντά στο δηλητηριώδες φρούτο έχουν το μικρότερο ευριστικό και άρα αυτοί είναι οι κόμβοι που εξετάζονται πρώτα. Όταν εμφανιστούν οι πρώτες καταστάσεις χωρίς δηλητηριώδες φρούτο τότε αυτόματα αυτές οι καταστάσεις έχουν μικρότερο ευριστικό σε σχέση με καταστάσεις όπου υπάρχει ακόμα δηλητηριώδες φρούτο. Μεταξύ των καταστάσεων χωρίς δηλητηριώδες φρούτο όμως, ο αλγόριθμος θα δώσει προτεραιότητα σε καταστάσεις όπου ο Pacman βρίσκεται όσο πιο κοντά στα περισσότερα φρούτα γίνεται, ακόμη κι αν χρειαστεί να αποφύγει να φάει ένα φρούτο(που θα ήταν λογικά η καλύτερη κίνηση). Έτσι το ευριστικό κριτήριο μπορεί να βελτιωθεί περαιτέρω αν προστεθεί και ένα κριτήριο για τον αριθμό των φρούτων που υπάρχουν σε κάθε κατάσταση.

Δένδρο Αναζήτησης BestFS:

Δίνεται και το δέντρο αναζήτησης BestFS για πιο εύκολη κατανόηση των αποτελεσμάτων. Ο αριθμός κάτω από την κάθε κατάσταση είναι το ευριστικό της κάθε μίας. Διαλέγουμε να επεκτείνουμε σε κάθε βήμα τον κόμβο με το μικρότερο ευριστικό.



6. Προσθέστε τη δυνατότητα παράλληλης παρακολούθησης της ουράς των μονοπατιών. Προσθέστε στον κώδικα αναζήτησης τη δυνατότητα επιλογής μεταξύ των μεθόδων που υλοποιήσατε.

Για να προσθέσουμε τη δυνατότητα παράλληλης παρακολούθησης της ουράς των μονοπατιών θα χρειαστεί να κάνουμε αρκετές αλλαγές. Πρώτα θα πρέπει να την αρχικοποιήσουμε και να μπορούμε να την επεκτείνουμε για κάθε μέθοδο όπως κάναμε με το μέτωπο.

Αλγόριθμοι DFS, BFS και BestFS με παρακολούθηση ουράς:

```
""" ------------------------------------------------------------------
---------
** initialization of queue
** Αρχικοποίηση ουράς
"""

def make_queue(state):
    return [[state]]    #Η ουρά είναι λίστα από κάθε μονοπάτι

""" ------------------------------------------------------------------
---------
**** expanding queue
**** επέκταση ουράς
"""

def extend_queue(queue, method):
    if method=='DFS':
        node=queue.pop(0)
        for child in find_children(node[-1]): # Εύρεση των παιδιών (απο
γόνων) της
        #τελευταίας κατάστασης κάθε μονοπατιού
            path=copy.deepcopy(node)
            path.append(child)
            queue.insert(0,path)    # Προσθέτουμε τα νέα μονοπάτια στην
ΑΡΧΗ του συνόλου

    elif method=='BFS':
        node=queue.pop(0)
        for child in find_children(node[-1]):
            path=copy.deepcopy(node)
            path.append(child)
            queue.append(path)  # Προσθέτουμε τα νέα μονοπάτια στο ΤΕΛΟ
Σ του συνόλου

    elif method=='BestFS':
        node = queue.pop(0)
        for child in find_children(node[-1]):
            path=copy.deepcopy(node)
            path.append(child)
            queue.append(path)
            queue.sort(key=lambda path: heuristic(path[-1]))    # Ταξιν
ομούμε με βάση
            # τις τελευταίες καταστάσεις του κάθε μονοπατιού με βάση το
            # ευριστικό κριτήριο
```

```
    print("\nQueue after expansion:")
    print_queue(queue)

    return queue
```

Εκτυπώνουμε το κάθε μονοπάτι με την συνάρτηση:

```
def print_queue(queue):
    for i, path in enumerate(queue):
            print(f"\nPath {i + 1}:")
            for state in path:
                print_state(state)   # Χρησιμοποιεί την print_state() π
ου φτιάξαμε πριν
```

Για να διαφοροποιήσουμε το παιχνίδι με παρακολούθηση ουράς φτιάχνουμε άλλη μέθοδο find_solution_with_queue:

```
""" ----------------------------------------------------------------
---------
**** Βασική αναδρομική συνάρτηση για δημιουργία δέντρου αναζήτησης (ανα
δρομική επέκταση δέντρου) με διαχείριση ουράς
"""
def find_solution_with_queue(front, queue, closed, method, step=0):
    if not front:  # Αν το μέτωπο είναι κενό, δεν βρέθηκε λύση
        print('No solution')

    elif front[0] in closed:  # Αν η πρώτη κατάσταση έχει ήδη εξεταστεί
        print("Step:", step)
        print("State Removed:")
        print_state(front[0])
        new_front = copy.deepcopy(front)
        new_front.pop(0)  # Αφαιρούμε την τρέχουσα κατάσταση και συνεχί
ζουμε
        new_queue = copy.deepcopy(queue)
        new_queue.pop(0)
        find_solution_with_queue(new_front, new_queue, closed, method,
step+1)

    elif is_goal_state(front[0]):  # Αν η τρέχουσα κατάσταση είναι ο στ
όχος
        print('Goal found in', step, 'steps!')
        print("Final State:")
        print_state(front[0])  # Εμφανίζει την τελική κατάσταση της πίσ
```

```
τας
        print("Full Path:")
        for state in queue[0]: # Εμφανίζει ολόκληρο το μονοπάτι που μας
πήγε
                                                    # στην τελκή κατ
άσταση
            print_state(state)

    else:
        print("\nStep:", step)
        closed.append(front[0])  # Προσθέτουμε την κατάσταση στη λίστα
των εξετασμένων
        front_copy = copy.deepcopy(front)
        front_children = expand_front(front_copy, method)
        print("Current State:")
        print_state(front[0])  # Εμφανίζει την τρέχουσα κατάσταση της π
ίστας
        queue_copy = copy.deepcopy(queue)
        queue_children = extend_queue(queue_copy, method)
        closed_copy = copy.deepcopy(closed)
        find_solution_with_queue(front_children, queue_children, closed
_copy, method, step+1)
```

Ένα σημαντικό σημείο που πρέπει να τονίστεί είναι πως για τον τελεστή μετάβασης destroy_fruit δημιουργείται σε τυχαίο σημείο ένα νέο φρούτο. Έτσι αυτό το σημείο μπορεί να είναι διαφορετικό για την κατάσταση που θα προστεθεί στο μέτωπο και για την κατάσταση που θα προστεθεί σε μονοπάτι της ουράς. Άρα μπορεί να προκληθεί λογικό σφάλμα και να βρεθεί διαφορετική ουρά από την λύση που ψάχνουμε. Για τον λόγο αυτό θα χρησιμοποιήσουμε μία σταθερή μεταβλητή seed για να σιγουρέψουμε πως θα λάβουν το ίδιο σημείο (την ίδια τιμή random). Έτσι θα κάνουμε τις ακόλουθες αλλαγές:

```
# Σταθερή τυχαία τιμή Global seed
GLOBAL_SEED = random.random()



def destroy_poison(state):
    pac_pos = find_pacman(state)
    if state[pac_pos][1] == 'd':  # Αν υπάρχει δηλητηριώδες φρούτο στο
κελί του Pacman
        state[pac_pos][1] = ' '  # Καταστρέφουμε το δηλητηριώδες φρούτο

        # Βρίσκουμε τυχαίο κενό κελί χωρίς τον Pacman για να τοποθετήσο
υμε το
        # νέο φαγώσιμο φρούτο
```

```
        random.seed(GLOBAL_SEED) #Προστέθηκε
        empty_cells = [i for i in range(len(state)) if state[i][0] !=
'p' and state[i][1] == ' ']
        if empty_cells:
            new_fruit_pos = random.choice(empty_cells)  # Τυχαία επιλογ
ή κενού κελιού
            # Κάναμε import random στην αρχή
            state[new_fruit_pos][1] = 'f'  # Τοποθετούμε το νέο φρούτο
        return state
    else:
        return None  # Δεν υπάρχει δηλητηριώδες φρούτο να καταστραφεί
```

Για να προσθέσουμε τη δυνατότητα επιλογής μεταξύ των μεθόδων αλλά και την δυνατότητα παρακολούθησης της ουράς αλλάζουμε την μέθοδο main:

```
""" ----------------------------------------------------------------
----------
** Κλήση Εκτέλεσης Κώδικα
"""

def main():

    initial_state=[[' ','d'],[' ','f'],['p',' '],[' ',' '],[' ','f'],['
',' ']]

    """ ------------------------------------------------------------
-------------
    **** Επιλογή Μεθόδου Αναζήτησης
    """

    print("Select the search method:")
    print("1. DFS (Depth-First Search)")
    print("2. BFS (Breadth-First Search)")
    print("3. Best-First Search")
    choice = int(input("Enter your choice (1, 2, or 3): "))

    if choice not in [1, 2, 3]:
        print("Invalid choice. Exiting program.")
        return

    methods = {1: 'DFS', 2: 'BFS', 3: 'BestFS'}
    method = methods[choice]
```

```
    """ ------------------------------------------------------------
-------------
    **** Χρήση Ουράς ή Όχι
    """
    print("Use queue-based approach?")
    print("1. Yes")
    print("2. No")
    queue_choice = int(input("Enter your choice (1, 2): "))

    """ ------------------------------------------------------------
-------------
    **** Έναρξη Αναζήτησης
    """

    print('Begin Searching:')

    if queue_choice==1:
        print(f"Using {method} with Queue.")
        find_solution_with_queue(make_front(initial_state), make_queue
(initial_state), [], method)
    else:
        print(f"Using {method} with Front.")
        find_solution(make_front(initial_state), [], method)



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

Δίνεται στο άλλο pdf ενδεικτικό τρέξιμο με το νέο κώδικα με παρακολούθηση ουράς.

7. Να παρουσιαστούν οι περιπτώσεις εξαντλητικού ελέγχου που θα χρησιμοποιήσετε για κάθε μέθοδο αναζήτησης και τα συμπεράσματα που θα βγάλετε από τη συγκριτική μελέτη των αποτελεσμάτων των δοκιμών σας μεταξύ διαφορετικών μεθόδων αναζήτησης.

Για τον έλεγχο θα εξετάσουμε διαφορετικές αρχικές καταστάσεις. Θα έχουμε:

- Κατάσταση 1: Ο Pacman ξεκινά στη μέση της πίστας με τα φρούτα στα άκρα, που είναι η εκφώνηση της άσκησης [' ','d'],[' ','f'],['p',' '],[' ',' '],[' ','f'],[' ',' ']
- Κατάσταση 2: Το δηλητηριώδες φρούτο είναι κοντά στον Pacman, αλλά τα φαγώσιμα είναι μακριά [' ','d'],['p',' '],[' ',' '],[' ',' '],[' ','f'],[' ','f']
- Κατάσταση 3: Το δηλητηριώδες φρούτο είναι όσο πιο μακριά από τον Pacman.

  ['p',' '],[' ','f'],[' ',' '],[' ','f'],[' ',' '],[' ','d']

Θα χρησιμοποιήσουμε την κάθε μέθοδο στις αρχικές αυτές καταστάσεις και θα συγκρίνουμε μεταξύ τους:

1. Τον Αριθμό Καταστάσεων που Εξετάστηκαν
2. Τον Χρόνο Υπολογισμού
3. Την Χρήση Μνήμης
4. Αν η μέθοδος βρήκε τη βέλτιστη λύση

Για να κάνουμε την σύκριση θα προσθέσουμε στον κώδικα συνάρτηση για την μέτρηση του χρόνου εκτέλεσης και την μέτρηση χρήση μνήμης:

```
""" ----------------------------------------------------------------
---------
**** Συνάρτηση Μέτρησης Χρόνου Εκτέλεσης και Χρήση Μνήμης
"""
def measure_performance(method, front, closed):
    import time
    import tracemalloc

    tracemalloc.start()  # Έναρξη παρακολούθησης μνήμης
    start_time = time.time()  # Έναρξη χρονικής μέτρησης

    find_solution(front, closed, method)

    end_time = time.time()  # Τέλος χρονικής μέτρησης
    current, peak = tracemalloc.get_traced_memory()  # Απόκτηση τρέχουσ
ας και
        # μέγιστης χρήσης μνήμης
    tracemalloc.stop()  # Τερματισμός παρακολούθησης μνήμης

    execution_time = end_time - start_time
    print(f"Method: {method}")
```

```
    print(f"Execution Time: {execution_time:.4f} seconds")
    print(f"Memory Usage: {peak / 10**6:.2f} MB (peak)")
```

Εκτελούμε τον κώδικα αλλάζοντας τα comments για κάθε περίπτωση που θέλουμε να εξετάσουμε:

```
def main():


    """"
    ----------------------------------------------------------------------
---------
    **** Εξέταση Χρόνου Εκτέλεσης και Χρήση Μνήμης
    """
    #initial_state_1 = [[' ','d'],[' ','f'],['p',' '],[' ',' '],[' '
,'f'],[' ',' ']]
    #initial_state_2 = [[' ','d'],['p',' '],[' ',' '],[' ',' '],[' '
,'f'],[' ','f']]
    #initial_state_3 = [['p',' '],[' ','f'],[' ',' '],[' ','f'],[' ','
'],[' ','d']]

    print("Αρχική Κατάσταση 1:")
    measure_performance('DFS', make_front(initial_state_1), [])
    #measure_performance('BFS', make_front(initial_state_1), [])
    #measure_performance('BestFS', make_front(initial_state_1), [])

    #print("Αρχική Κατάσταση 2:")
    #measure_performance('DFS', make_front(initial_state_2), [])
    #measure_performance('BFS', make_front(initial_state_2), [])
    #measure_performance('BestFS', make_front(initial_state_2), [])

    #print("Αρχική Κατάσταση 3:")
    #measure_performance('DFS', make_front(initial_state_3), [])
    #measure_performance('BFS', make_front(initial_state_3), [])
    #measure_performance('BestFS', make_front(initial_state_3), [])


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

Αποτελέσματα:

Μέθοδος	Αρχική Κατάσταση	Αριθμός Καταστάσεων (Βήματα)	Χρόνος Εκτέλεσης (sec)	Μνήμη (MB - Μέγιστη)	Βέλτιστη Λύση

DFS	1	25	0.0347	0.27	Όχι
BFS	1	104	0.3061	2.40	Ναι
Best-First	1	17	0.0509	0.23	Όχι
DFS	2	19	0.0272	0.21	Όχι
BFS	2	83	0.1946	1.57	Ναι
Best-First	2	14	0.0361	0.17	Όχι
DFS	3	16	0.0219	0.18	Όχι
BFS	3	99	0.2859	2.22	Ναι
Best-First	3	17	0.0560	0.25	Όχι

Συμπεράσματα:

DFS (Πρώτα σε Βάθος)

- **Πλεονεκτήματα**:
  - Μικρός χρόνος εκτέλεσης - Μικρός αριθμός βημάτων
  - Χαμηλή χρήση μνήμης, καθώς αποθηκεύει μόνο το μονοπάτι που εξετάζεται.
  - Αποτελεσματικό για εύρεση λύσεων αν ο στόχος είναι σε μεγάλο βάθος.
- **Μειονεκτήματα**:
  - Μπορεί να χαθεί σε άπειρο βάθος αν δεν υπάρχει λύση. Έτσι είναι καλό για το πρόβλημά μας αλλά όχι για άλλα προβλήματα που μπορεί να έχουν πολλές “αδιέξοδους”.
  - Δε διασφαλίζει τη βέλτιστη λύση.

BFS (Πρώτα σε Πλάτος)

- **Πλεονεκτήματα**:
  - Εγγυάται τη βέλτιστη λύση.
  - Εξετάζει όλες τις καταστάσεις στο ίδιο επίπεδο πριν προχωρήσει σε μεγαλύτερο βάθος.
- **Μειονεκτήματα**:
  - Μεγάλο χρόνο εκτέλεσης - Μεγάλος αριθμός βημάτων για προβλήματα που ο στόχος είναι σε μεγάλο βάθος.
  - Υψηλή χρήση μνήμης, καθώς διατηρεί όλα τα μονοπάτια στο μέτωπο.

Best-First Search

- **Πλεονεκτήματα**:
  - Εξετάζει πρώτα τις πιο υποσχόμενες καταστάσεις βάσει ευριστικής συνάρτησης.
  - Παρόμοιο αριθμό βημάτων και χρόνο εκτέλεσης με το DFS.
- **Μειονεκτήματα**:

- Στο τέλος, εξαρτάται από την ευριστική: αν η ευριστική είναι κακή, μπορεί να οδηγήσει σε λιγότερο αποδοτική αναζήτηση.