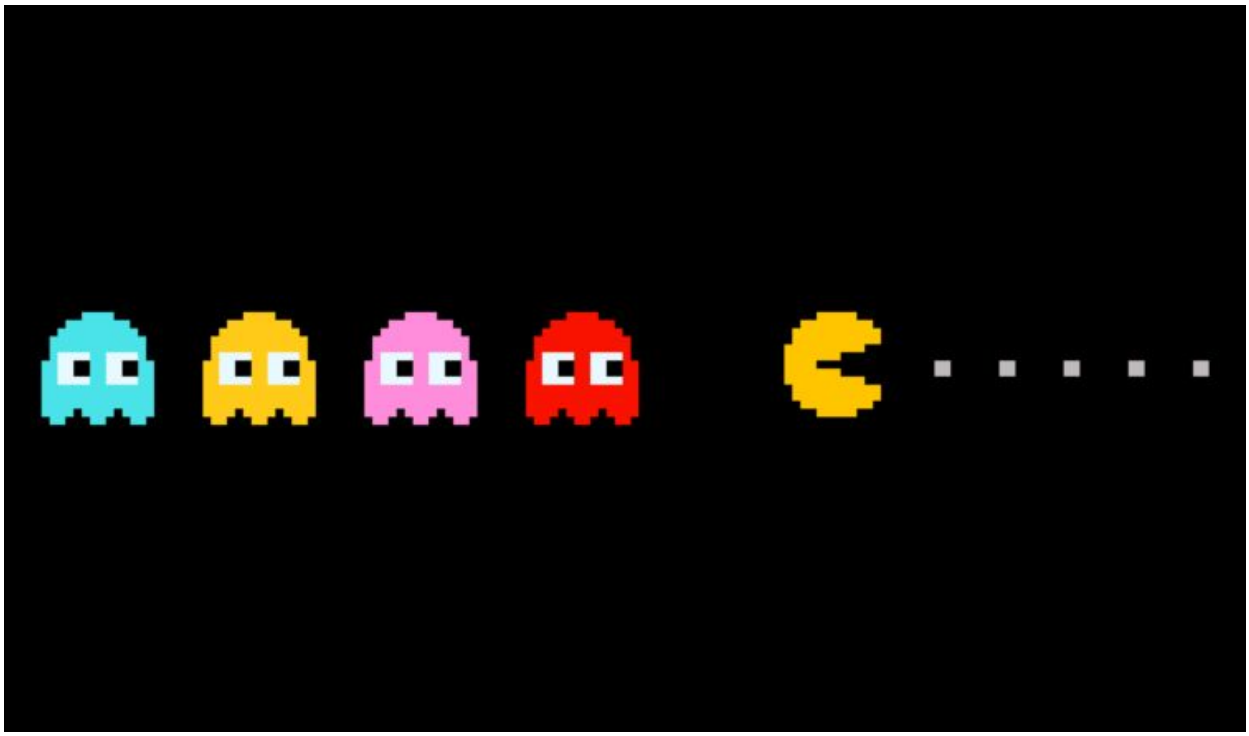


Pacman

Τεχνητή Νοημοσύνη Ασκ 1

Νικόλα Σιάκας 171011

Περιεχόμενα

Ο κόσμος του Pacman

Αρχική κατάσταση

Ο κόσμος αποτελείται από n γραμμές και m στήλες. Στον κόσμο υπάρχει ένας μόνο pacman, κάποια φρούτα και κάποιοι τοίχοι. Ο pacman δεν μπορεί να μετακινηθεί σε κελί στο οποίο βρίσκεται τοίχος.

Στόχος

Ο στόχος του pacman είναι να φάει όλα τα φρούτα στον κόσμο. Αποδεκτή κατάσταση θεωρείται κάθε κατάσταση στην οποία δεν υπάρχουν φρούτα

Τελεστές

Οι κινήσεις που μπορεί να κάνει ο pacman ειναι οι εξεις:

1. Μετακίνηση προς τα δεξιά
2. Μετακίνηση προς τα αριστερά
3. Μετακίνηση προς τα πάνω
4. Μετακίνηση προς τα κάτω
5. Φάγωμα φρούτου

Ο pacman μπορεί να εκτελέσει έναν μόνο τελεστή σε κάθε κίνηση του

Pacman abstract

Περιγραφή

Στην abstract αναπαράσταση του προβλήματος του pacman δεν κρατάμε μια λίστα από λίστες από λίστες για να αναπαραστήσουμε έναν κόσμο $n \, x \, m$ αλλά κρατάμε μόνο τις συντεταγμένες που μας ενδιαφέρουν ετσι η εύρεση της λυσης είναι πιο γρήγορη γιατί δεν ψάχνουμε καθε φορά σε όλο τον κόσμο για να βρούμε τον pacman

Κλαση State

Στο αρχείο `world.py` βρίσκετε η κλάση `State`. Η State αναπαραστα την κατάσταση που βρίσκεται ο pacman. Η κλαση κρατάει σε μια λίστα τις συντεταγμένες του pacman, το πρώτο στοιχείο της λίστας είναι ο αριθμός της γραμμής στην οποία βρίσκεται και το δεύτερο στοιχείο είναι ο αριθμός της στήλης στην οποία βρίσκεται ο pacman. Η κλάση επίσης κρατάει σε δύο ξεχωριστές λίστες τις συντεταγμένες των τοίχων και των φρούτων σαν tuples[1] από δύο στοιχεία, τον αριθμός της γραμμής και τον αριθμος της στηλης στην οποία βρίσκεται αντίστοιχα

Attributes της State

Class Attributes[2]:

```
__ROW_INDEX = 0

__COL_INDEX = 1
```

Βοηθητικές μεταβλητές για να κάνουμε index την γραμμή και την στήλη στην λίστα

[1] Tuples γιατί οι τοποθεσίες των φρούτων και των τοίχων δεν αλλάζει σε αντιθεση με τον pacman που μετακινείται
[2] Class attributes είναι οι μεταβλητές της κλάσης, έχουν ίδια τιμή για όλα τα αντικείμενα της κλάσης και μπορούμε να τα προσπελάσουμε χωρίς κάποιο αντικείμενο, κατευθείαν από την κλάση

Private Instance Attributes[3]:

```
self.__rows

Self.__columns
```

Στην `__rows` αποθηκεύεται ο αριθμος των γραμμων του κόσμου και στην `__columns` ο αριθμός των στηλών.

Public Instance Attributes:

```
self.pacman_location = []

self.walls_locations = []

self.fruits_locations = []
```

- pacman_location: Μια λίστα στην οποία κρατιέται η θέση του pacman στην μορφη **`[γραμμή, στήλη]`**
- walls_locations: Μια λίστα από tuples για τις συντεταγμένες των τοίχων. Καθε tuple είναι της μορφής **`(γραμμή, στήλη)`** για καθε τοίχο στον κόσμο
- fruits_location: Μια λίστα από tuples για τις συντεταγμένες των φρούτων. Καθε tuple είναι της μορφής **`(γραμμή, στήλη)`** για καθε φρουτο στον κόσμο

Methods της State

`__init__(self, rows: int, columns: int, num_of_fruits: int = -1, num_of_walls: int = -1)`

Args:

- `rows`: Ο αριθμός των γραμμών του κόσμου
- `columns`: Ο αριθμός των στηλών του κόσμου
- `num_of_fruits`: Ο αριθμος των φρούτων στον κόσμο. Αν δοθεί μη έγκυρη τιμή τότε θα παρει μια τυχαία μεταξύ 1 και rows*columns//2
- `num_of_walls`: Ο αριθμος των τοίχων στον κόσμο. Αν δοθεί μη έγκυρη τιμή τότε θα παρει μια τυχαία μεταξύ 0 και rows*columns//3

[3] Στην python για να κάνουμε ένα attribute private βάζουμε __ (διπλό underscore) στην αρχή του ονόματος του

Περιγραφή:

Ο Constructor της κλασης State. Για αρχή αποθηκεύει τα rows και columns στα αντίστοιχα private attributes(__rows, __columns) και βάζει τον pacman σε μια τυχαία θέση στον κόσμο. Μετά βρίσκει τις τυχαίες τοποθεσίες για τους τοίχους με κριτήρια να μην υπαρχει τοιχος σε αυτην την θέση και να μην είναι ο pacman σε αυτην την θέση. Στην συνέχεια βρίσκει τις τυχαίες θέσεις για τα φρούτα με κριτήρια να μην υπάρχει ήδη φρουτο εκει και να μην ειναι τοιχος σε αυτη την θέση.

`_canEat(self) -> bool`

Protected[4] method που ελέγχει αν ο pacman βρίσκεται σε θέση με κάποιο φρούτο. Αν ναι τοτε επιστρέφει `True`.

`_canMoveLeft(self) -> bool`

Protected method που ελέγχει αν ο pacman μπορεί να μετακινηθεί προς τα αριστερά. Ο pacman μπορεί να μετακινηθεί προς τα αριστερά αν η στήλη στην οποια βρίσκεται είναι μεγαλύτερη του 0[5] και αν στην αριστερή θέση δεν υπάρχει τοίχος.

`_canMoveRight(self) -> bool`

Protected method που ελέγχει αν ο pacman μπορεί να μετακινηθεί προς τα δεξιά. Ο pacman μπορεί να μετακινηθεί προς τα δεξιά αν η στήλη στην οποία βρίσκεται είναι μικρότερη του __columns-1[6] και αν στην δεξιά θέση δεν υπάρχει τοίχος.

`_canMoveUp(self) -> bool`

Protected method που ελέγχει αν ο pacman μπορεί να μετακινηθεί προς τα πάνω. Ο pacman μπορεί να μετακινηθεί προς τα πάνω αν η γραμμή στην οποία βρίσκεται είναι μεγαλύτερη από το 0 και αν στην πάνω θέση δεν υπάρχει τοίχος.

[4] Στην python οι μεταβλητές και οι συναρτήσεις/μέθοδοι που το όνομα του ξεκινά με _ (underscore) θεωρούνται protected
[5] Η πιο αριστερά στήλη είναι η 0
[6] Η πιο δεξια στήλη είναι η __colums-1

`_canMoveDown(self) -> bool`

Protected method που ελέγχει αν ο pacman μπορεί να μετακινηθεί προς τα κάτω. Ο pacman μπορεί να μετακινηθεί προς τα κάτω αν η γραμμή στην οποία βρίσκεται είναι μικρότερη από __rows-1[7] και αν στην πάνω θέση δεν υπάρχει τοίχος.

`eat(self)`

Αν ο pacman βρίσκεται σε θέση με κάποιο φρούτο τότε βγάζουμε από την λίστα με τις συντεταγμένες των φρούτων το tuple με αυτες τις συντεταγμένες.

```
self.fruits_locations.pop(self.fruits_locations.index(tuple(self
.pacman_location)))
```

Ξέρουμε ότι το tuple self.pacman_location βρίσκεται στην λίστα με τις συντεταγμένες των φρούτων και έτσι με το `index` παίρνουμε την θέση στην οποια βρίσκεται το tuple με αυτες τις συντεταγμένες και το κανουμε `pop`. Καλείται η _canEat για να ελέγξει αν ο pacman βρίσκεται στην ίδια θέση με κάποιο φρούτο.

`moveLeft(self)`

Αν η _canMoveLeft επιστρέψει `True` τότε αλλάζει τις συντεταγμένες του pacman σε **`[γραμμή, στήλη-1]`**.

`moveRight(self)`

Αν η _canMoveRight επιστρέψει `True` τότε αλλάζει τις συντεταγμένες του pacman σε **`[γραμμή, στήλη+1]`**.

`moveUp(self)`

Αν η _canMoveUp επιστρέψει `True` τότε αλλάζει τις συντεταγμένες του pacman σε **`[γραμμή-1, στήλη]`**.

[7] Μετράμε τις γραμμές από το 0 μεχρι __rows-1

`moveDown(self)`

Αν η _canMoveDown επιστρέψει `True` τότε αλλάζει τις συντεταγμένες του pacman σε **`[γραμμή+1, στήλη]`**.

`reachedGoal(self) -> bool`

Επιστρέφει `True` αν η κατάσταση είναι τελική. Τελική είναι οποιαδήποτε κατάσταση στην οποία η λίστα `self.fruits_locations` είναι άδεια.

`distanceToClosestFruit(self) -> int`

Επιστρέφει τον αριθμό των βημάτων που πρέπει να κάνει ο pacman για να φτάσει στο κοντινότερο κελί με φρούτο. Η απόσταση μετριέται με βάση των παρακάτω τύπο:

$$distance = |fruit[\text{γραμμή}] - pacman[\text{γραμμή}]| + |fruit[\text{στήλη}] - pacman[\text{στήλη}]|$$

Χρησιμοποιούμε τον παραπάνω τύπο για να βρούμε την απόσταση του pacman από το κάθε φρούτο και επιστρέφουμε την μικρότερη τιμή. Αν δεν υπάρχουν φρούτα επιστρέφουμε -1. Σημείωση: προς το παρόν αγνοούνται οι τοίχοι στην εκτίμηση της απόστασης.

`__eq__(self, other) -> bool`

Overload το `==`, καλείται όταν συγκρίνουμε αν δύο αντικείμενα τύπου `State` είναι ίσα και όταν χρησιμοποιούμε το `in` για να βρούμε αν μια κατάσταση βρίσκεται μέσα σε κάποια λίστα από αντικείμενα τύπου `State`. Δυο καταστάσεις θεωρούνται ίσες αν:

1. Το μέγεθος του κόσμου είναι ίδιο
2. Ο pacman βρίσκεται στην ίδια θέση
3. Έχουν ίδιο αριθμό φρούτων
4. Τα φρούτα τους βρίσκονται στις ίδιες θέσεις
5. Έχουν τον ίδιο αριθμό τοίχων
6. Οι τοίχοι τους βρίσκονται στις ίδιες θέσεις

Επιστρέφει `True` αν ισχύουν τα παραπάνω.

`__str__(self) -> str`

Η `__str__` καλείται όταν κάνουμε `print` κάποιο αντικείμενο της κλάσης `State`.

Επιστρέφει ένα `string` που αναπαριστά το state με την παρακάτω μορφή.

Το **`p`** αναπαριστά τον pacman.

Τα **`f`** αναπαριστούν τα φρούτα.

Τα **`w`** αναπαριστούν τους τοίχους.

Τα κενά τετράγωνα αναπαριστούν τους διαδρόμους στους οποίους μπορεί να μετακινηθεί ο pacman.

```
+---+---+---+
| f | w |   |
+---+---+---+
|   | f | w |
+---+---+---+
| w |p f|   |
+---+---+---+
```

`__deepcopy__(self, memo)`

Override την `deepcopy`. Όταν καλείται η `deepcopy` με παράμετρο ένα αντικείμενο τύπου `State` ή κάποιο iterable object που έχει μέσα του κάποιο αντικείμενο τυπου `State` ακόμα και όταν είναι nested μέσα σε άλλα iterable object, εκτελείται αυτή η μέθοδος. Είναι το standard override της deepcopy στο οποίο κρατάμε ένα dictionary το οποιο έχει σαν κλειδί το `id` κάποιου object και σαν τιμή ένα αντίγραφο του object αυτού ώστε να ξέρει τι έχει αντιγράψει. Η μέθοδος αντιγράφει ολα τα attributes της `State` και κάνει set τις τιμές στο καινούργιο αντικείμενο `State`.

Search

Περιγραφή

Το main αρχείο του project είναι το `search.py`. Το αρχείο αυτό υλοποιεί τους αλγόριθμους αναζήτησης που χρησιμοποιούμε για να βρούμε λύση στο πρόβλημα του pacman. Προς το παρόν έχουν αναπτυχθεί δύο τυφλοί αλγόριθμοι αναζήτησης, ο DFS (Depth First Search - Πρώτα σε βάθος αναζήτηση) και ο BFS (Breadth First Search - Πρώτα σε πλάτος αναζήτηση) και ένας ευριστικός αλγόριθμος, ο Best-First.

Imports

```
from world import State

from copy import deepcopy

from typing import List, Tuple

import time
```

`State`: Η κλάση που αναπαριστά το state του κόσμου.

`deepcopy`: Η συνάρτηση deepcopy χρησιμοποιείται για να αντιγράψουμε τις λίστες.

`typing`: Το module αυτό χρησιμοποιείται για δώσει ένα hint στον προγραμματιστή για το τι τύπου είναι οι παράμετροι που περιμένει μια συνάρτηση και τι τύπο θα του επιστρέψει μετά την εκτέλεση της. Κάνοντας `import` τις δομές `List` και `Tuple` μπορούμε να βάζουμε hints και για το τι θα περιέχει η λίστα/πλειάδα[8].

`time`: Το κάνουμε import για να μετρήσουμε πόσο χρόνο πήρε η αναζήτηση.

[8] tuple

Συναρτήσεις

`main()`

Αρχικοποιεί τις μεταβλητές `rows`, `columns`, `num_of_fruits`, `num_of_walls`, `method` καλώντας την συνάρτηση `menu`. Η μεταβλητή `method` είναι τύπου `str` και δηλώνει τον αλγόριθμο αναζήτησης που θέλει να χρησιμοποιήσει ο χρήστης(DFS, BFS, Best-First). Μετά δημιουργεί το αντικείμενο `initial_state` τύπου `State`, χρησιμοποιώντας τις παραπάνω μεταβλητές σαν παραμέτρους, το οποίο αντιπροσωπεύει την αρχική κατάσταση του κόσμου. Στην συνέχεια αφού εμφανίσει κάποια μηνύματα ξεκινάει την χρονομέτρηση[9] και καλεί την αναδρομική συνάρτηση `findSolution`.

`menu() -> Tuple[int, int, int, int, str]`

Διαβάζει από το πληκτρολόγιο τις τιμές για τις παραμέτρους του κόσμου. Γίνονται έλεγχοι για την εγκυρότητα των τιμών για τις γραμμές και τις στήλες του κόσμου, για να είναι έγκυρες πρέπει να είναι και οι δύο θετικοί και το γινόμενο τους να είναι μεγαλύτερο από δύο δηλαδή να έχει τουλάχιστον δύο κελιά, δεν γίνεται για τους αριθμους των φρούτων και των τοίχων γιατί ελέγχονται από τον κατασκευαστή της `State`. Για την μέθοδο αναζήτησης υπάρχει ένα `dict` με κλειδί έναν αύξοντα αριθμό και τιμή το όνομα του αλγορίθμου, από τον χρήστη ζητείται ο α/α και επιστρέφεται η τιμή που αντιστοιχεί σε αυτον τον α/α, αν ο α/α δεν είναι έγκυρος επιστρέφεται το `str` DFS ως προκαθορισμένος αλγόριθμος.

```
findSolution(front: List[State], queue: List[List[State]],
closed: List[State] = [], method: str = 'DFS')
```

Args:

- `front`: Μια λίστα από αντικείμενα τύπου `State`. Το μέτωπο αναζήτησης
- `queue`: Μια λίστα από λίστες από αντικείμενα τύπου `State`. Σε αυτήν την λίστα περιέχονται τα μονοπάτια[10] που οδηγούν σε κάποια κατάσταση. Το στοιχείο `i` της λίστας είναι το μονοπάτι για το `i` στοιχείο της λίστας `front`
- `Closed`: Μια λίστα από αντικείμενα τύπου `State` τα οποία έχουμε εξετάσει

[9] Η χρονομέτρηση υλοποιείται αποθηκεύοντας δευτερόλεπτα που έχει κάνει αυτή η διεργασία μέσα στην CPU μέχρι τώρα και αφαιρώντας τα από τα δευτερόλεπτα που θα έχουν περάσει όταν τελειώσει η αναζήτηση
[10] Μια λίστα από αντικείμενα τύπου `State` που οδηγούν σε μια κατάσταση

- `method`: Η μέθοδος αναζήτησης που θα χρησιμοποιηθεί

Περιγραφή:

Υπάρχουν 4 καταστάσεις στις οποίες μπορεί να βρίσκεται το μέτωπο αναζήτησης

1. Άδειο μέτωπο. Τερματική περίπτωση, όταν το μέτωπο είναι άδειο δεν υπάρχει λύση για τον κόσμο
2. Η κεφαλή[11] του μετώπου είναι τελική κατάσταση. Τερματική περίπτωση, βρέθηκε λύση για το πρόβλημα και βρίσκεται στην κεφαλή του μετώπου και η διαδρομή[12] για να φτάσουμε σε αυτην την κατάσταση βρίσκεται στην κεφαλή της λίστας `queue`
3. Η κατάσταση που βρίσκεται στην κεφαλή του μετώπου έχει αναλυθεί οπότε την βγάζουμε από το μέτωπο
4. Στην κεφαλή του μετώπου βρίσκεται μια κατάσταση η οποία δεν είναι τελική και δεν έχει αναλυθεί

Άμα βρισκόμαστε στην περίπτωση 2 εμφανίζουμε τις κινήσεις που εκτέλεσε ο αλγόριθμος για να κάνει τον pacman να φάει όλα τα φρούτα. Στην 3η περίπτωση βγάζουμε από το μέτωπο και απο την ουρά τις κεφαλές τους, γιατί αυτή η κατάσταση του κόσμου έχει εξεταστεί, και ξανά καλούμε την συνάρτηση με το καινούργιο μετωπο και την καινουργια ουρα σαν ορίσματα. Τέλος αν βρισκεται στην 4η περίπτωση θα προσθέσουμε την κατάσταση στις κλειστές και θα καλέσει την `expandFront` και την `expandQueue`, οι οποίες θα τροποποιήσουν το μέτωπο και τις ουρές ανάλογα με τον αλγόριθμο αναζήτησης που διάλεξε ο χρήστης, και μετα θα καλέσει τον εαυτό της με ορίσματα το καινούργιο μέτωπο, την καινούργια `queue` και την καινούργια λίστα με τις κλειστές καταστάσεις.

```
expandFront(front: List[State], method: str = 'DFS')
```

Args:

- `front`: Μια λίστα από αντικείμενα τύπου `State`. Το μέτωπο αναζήτησης
- `method`: Η μέθοδος αναζήτησης που θα χρησιμοποιηθεί

Περιγραφή:

Βγάζει από το μέτωπο αναζήτησης το πρώτο στοιχείο και βάζει τις καταστάσεις παιδιά του καλώντας την `findChildren`. Η θέση στην οποία θα βάλει τις καταστάσεις εξαρτάται από την `method`, αναλυτικά αν η μέθοδος είναι DFS τότε θα βάλει τα παιδιά στην αρχή του μετώπου αντίθετα από την BFS μέθοδο η οποία θα τα βάλει στο τέλος του μετώπου , τέλος όταν χρησιμοποιείται η μέθοδος Best-First η συνάρτηση θα ταξινομήσει

[11] Το στοιχείο 0 της λίστας front
[12] Μια διαδρομή ενδεχομένως να υπάρχουν και άλλες

το νέο μέτωπο αναζήτησης με βάση ποια κατάσταση είναι πιο κοντά στην τελική, τα κριτήρια που έχουμε επιλέξει για την ταξινόμηση είναι το πόσα φρούτα έχουν μείνει στην κατάσταση και ως δεύτερο κριτήριο το πόσα βήματα απέχει ο pacman από το πιο κοντινό του φρούτο. Οι αλγόριθμοι αναζήτησης αναλύονται περαιτέρω στην σελίδα 15.

`expandQueue(queue: List[List[State]], method: str = 'DFS')`

Args:

- `queue`: Μια λίστα από λίστες από αντικείμενα τύπου `State`. Σε αυτήν την λίστα περιέχονται τα μονοπάτια που οδηγούν στην κατάσταση που βρίσκεται στο τέλος του κάθε μονοπατιού.
- `method`: Η μέθοδος αναζήτησης που θα χρησιμοποιηθεί

Περιγραφή:

Βγάζει από την λίστα το πρώτο μονοπάτι[13] και βρίσκει τα επόμενα βήματα που μπορεί να κάνει το μονοπάτι καλώντας την `findChildren`. Για κάθε παιδί που έχει αυτό το μονοπάτι[14] αντιγράφει το μονοπάτι και του προσθέτει στο τέλος την κατάσταση παιδί και μετά βάζει το καινούργιο μονοπάτι στην `queue`. Τα καινούργια μονοπάτια προσθέτονται στην `queue` ανάλογα με την μέθοδο αναζήτησης που έχει επιλέξει ο χρήστης, στην μέθοδο DFS τα καινούργια μονοπάτια προστίθονται στην αρχή της `queue` ενώ στην BFS μέθοδο στο τέλος της `queue` και στην μέθοδο Best-First γίνεται ταξινόμηση με βάση ποιό μονοπάτι οδηγεί στην “καλύτερη” κατάσταση με βάση σε ποιά κατάσταση έχουν απομείνει λιγότερα φρούτα και ως δεύτερο κριτήριο σε ποιά κατάσταση ο pacman βρίσκεται πιο κοντά σε κάποιο φρούτο.

`findChildren(state: State) -> list`

Η συνάρτηση αυτή παίρνει ως παράμετρο μια κατάσταση και επιστρέφει μια λίστα από τις καταστάσεις παιδιά της. Οι καταστάσεις παιδιά είναι οι καταστάσεις οι οποίες δημιουργούνται αν εφαρμόσουμε κάποιον τελεστή. Η συνάρτηση επιστρέφει μόνο τα παιδιά στα οποία εφαρμόστηκε κάποιος τελεστής με επιτυχία, δηλαδή να η αρχική κατάσταση `state` πρέπει να έχει αλλάξει για να θεωρείται ότι ο τελεστής εκτελέστηκε με επιτυχία.

[13] Μια λίστα από αντικείμενα τύπου `State`
[14] Τα παιδιά του μονοπατιού είναι οι καταστάσεις παιδιά του τελευταίου στοιχείου του μονοπατιού

`printFront(front: List[State])`

Βοηθητική συνάρτηση για να εκτυπώσουμε το μέτωπο αναζήτησης σε μια πιο ευανάγνωστη μορφή.

`printQueue(queue: List[List[State]])`

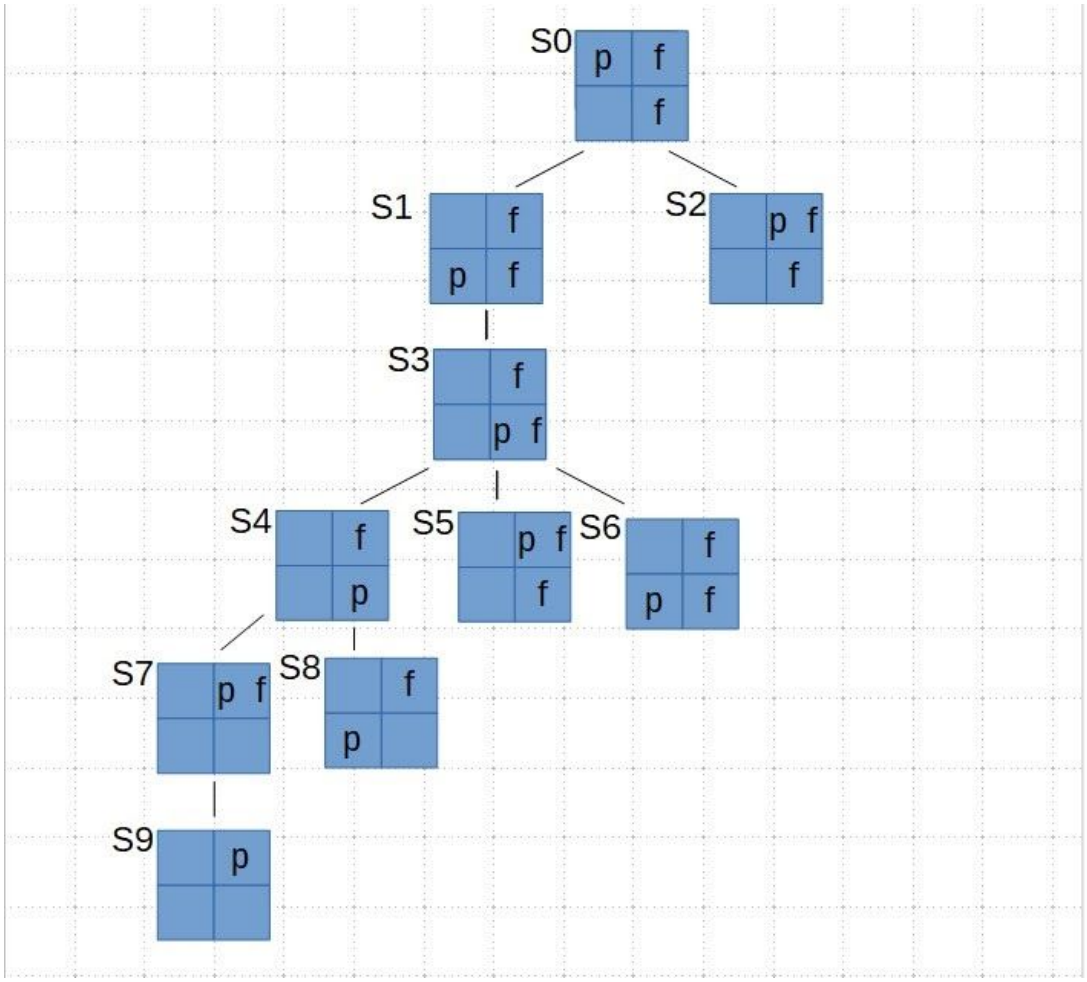
Βοηθητική συνάρτηση για να εκτυπώσουμε τα μονοπάτια που οδηγούν στις καταστάσεις οι οποίες βρίσκονται στο μέτωπο αναζήτησης σε μια πιο ευανάγνωστη μορφή.

Αλγόριθμοι αναζήτησης

Στην συνέχεια θα αναλυθούν οι 3 αλγόριθμοι αναζήτησης που χρησιμοποιούμε για να βρούμε λύση στο πρόβλημα του pacman. Οι αλγόριθμοι θα αναλυθούν με την ίδια αρχική κατάσταση η οποία συμβολίζεται ως S0.

DFS

Σε αυτή την μέθοδο αναζήτησης οι καταστάσεις παιδιά προστίθονται στην αρχή του μετώπου. Πιο αναλυτικά, βγάζουμε από το μέτωπο την πρώτη κατάσταση και αν δεν έχει αναλυθεί[15] τότε βρίσκονται οι καταστάσεις παιδιά του και προστίθονται στην αρχή του μετώπου



[15] Μια κατάσταση θεωρείται ότι έχει αναλυθεί όταν έχουμε βρει τα παιδιά της και τα έχουμε προσθέσει στο μέτωπο

Μέτωπο = [S0]

Μέτωπο = [S1, S2]

Μέτωπο = [S3, S2]

Μέτωπο = [S4, S5, S6, S2]

Μέτωπο = [S7, S8, S5, S6, S2]

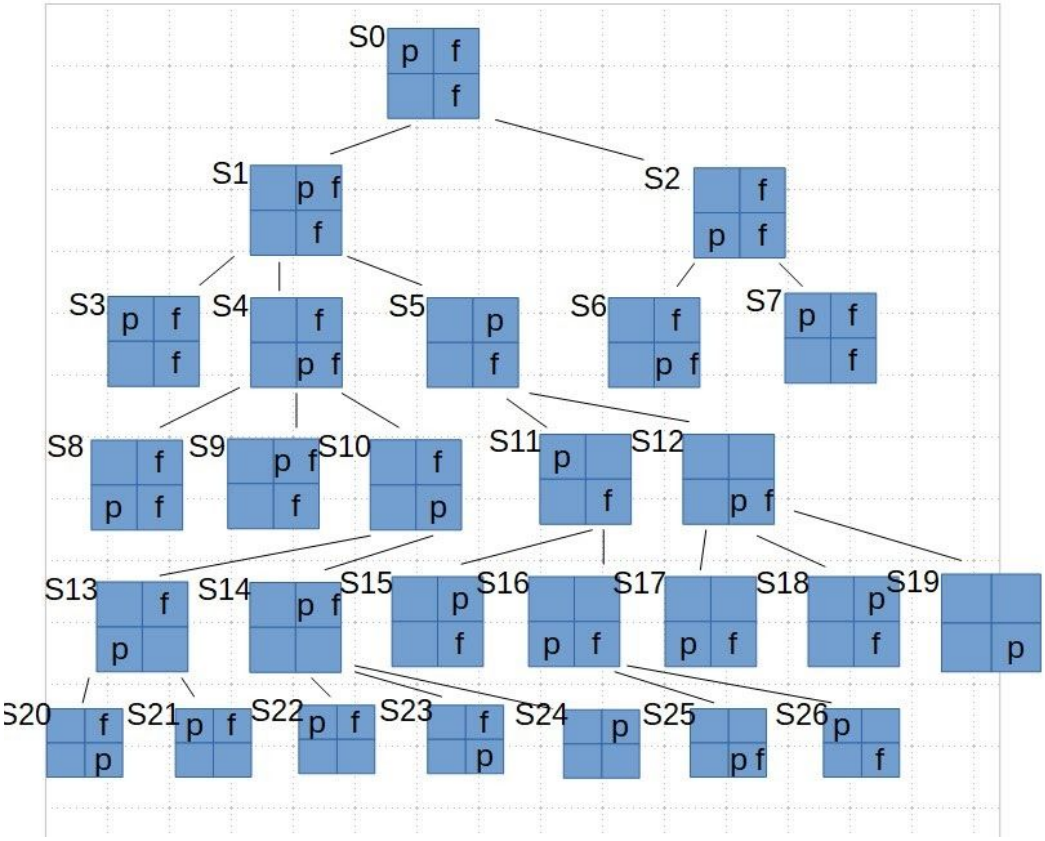
Μέτωπο = [S9, S8, S5, S6, S2]

Τέλος γιατί στην κεφαλή του μετώπου βρίσκεται μια τελική κατάσταση.

Τελικό μονοπάτι = [S0, S1, S3, S4, S7, S9]

BFS

Στην BFS μέθοδο οι καταστάσεις παιδιά προστίθονται στο τέλος του μετώπου. Πιο αναλυτικά, βγάζουμε από το μέτωπο την πρώτη κατάσταση και αν δεν έχει αναλυθεί τότε βρίσκονται οι καταστάσεις παιδιά του και προστίθονται στο τέλος του μετώπου



Μέτωπο = [S0]

Μέτωπο = [S1, S2]

Μέτωπο = [S2, S3, S4, S5]

Μέτωπο = [S4, S5, S6, S7]. Η S3 βγήκε από το μέτωπο γιατί έχει ήδη αναλυθεί, είναι ίδια με την S0.

Μέτωπο = [S5, S6, S7, S8, S9, S10]

Μέτωπο = [S10, S11, S12]. Η S6 βγήκε γιατί είναι ίδια με την S4 η οποία έχει αναλυθεί, η S7 είναι ίδια με την S0 οπότε και αυτή βγαίνει από το μέτωπο, παρόμοια η S8 και S9 βγήκαν γιατί είναι ίδιες με τις S2 και S1 αντίστοιχα.

Μέτωπο = [S11, S12, S13, S14]

Μέτωπο = [S12, S13, S14, S15, S16]

Μέτωπο = [S13, S14, S15, S16, S17, S18, S19]

Μέτωπο = [S14, S15, S16, S17, S18, S19, S20, S21]

Μέτωπο = [S16, S17, S18, S19, S20, S21, S22, S23, S24]. Η S15 βγήκε από το μέτωπο γιατί είναι ίδια με την S5 η οποία έχει αναλυθεί.

Μέτωπο = [S19, S20, S21, S22, S23, S24, S25, S26]. Οι S17 και S18 βγήκαν από το μέτωπο γιατί είναι ίδιες με τις S16 και S5 αντίστοιχα οι οποίες έχουν αναλυθεί.

Τέλος γιατί στην κεφαλή του μετώπου βρίσκεται μια τελική κατάσταση.

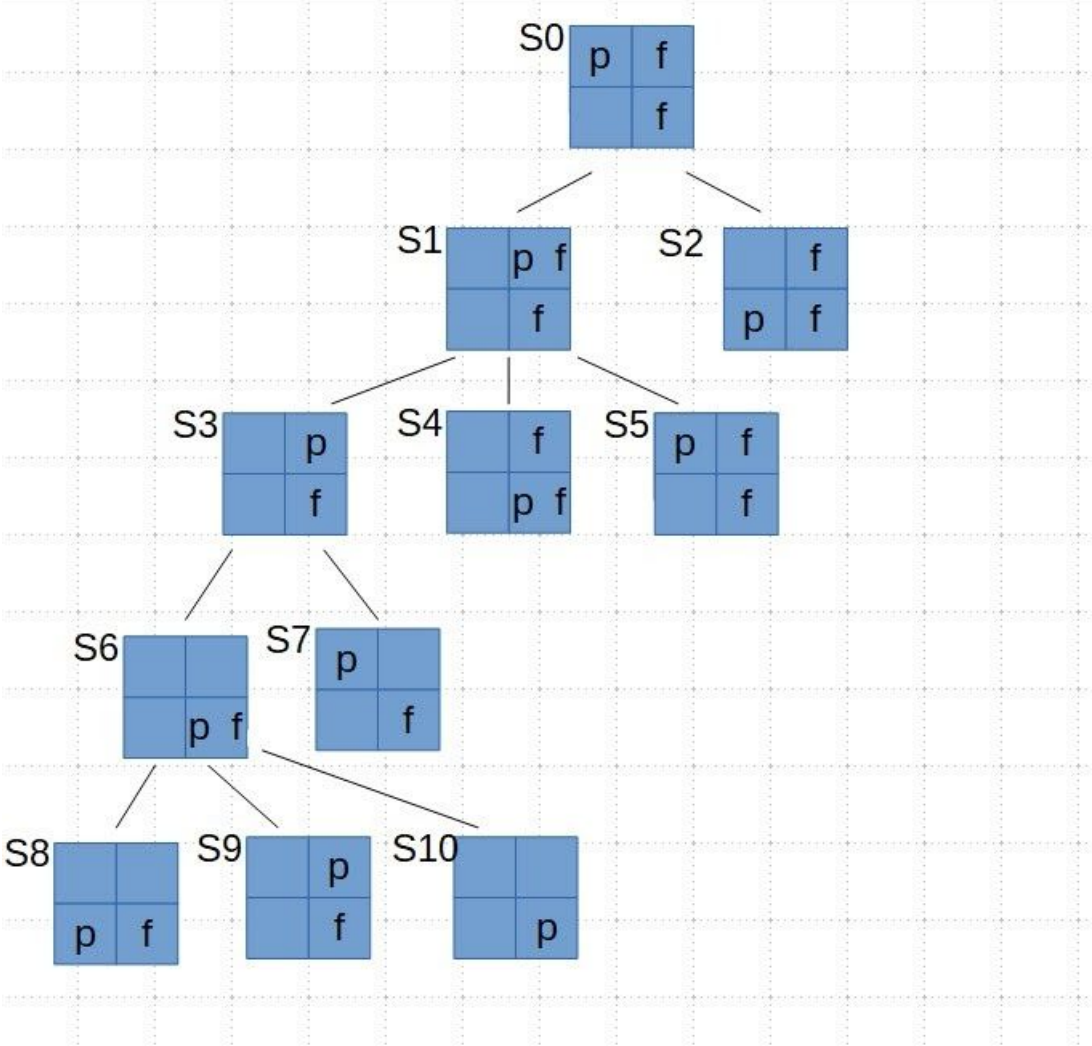
Τελικό μονοπάτι = [S0, S1, S5, S12, S19]

Όπως βλέπουμε με τον BFS βρήκαμε μονοπάτι που οδηγεί σε τελική κατάσταση με λιγότερα βήματα από ότι ο DFS, όμως χρειάζεται περισσότερος χρόνος για να βρεθεί λύση

Best-First

Στην Best-First αναζήτηση οι καταστάσεις παιδιά προστίθονται στο μέτωπο[16] και μετά γίνεται μια ταξινόμηση με βάση κάποιο ευριστικό κριτήριο. Στην δική μας λύσει τα κριτήρια είναι δύο, το πλήθος των φρούτων στον κόσμο καταστάσεις με λιγότερα φρούτα θα βρίσκονται στην αρχή του μετώπου μετά την ταξινόμηση και επιπλέον αν οι καταστάσεις έχουν τον ίδιο αριθμό από φρούτα στον κόσμο τότε κοιτάμε σε ποιά κατάσταση ο pacman βρίσκεται πιο κοντά σε φρούτο.

[16] Δεν έχει σημασία η θέση, στην δική μας εκδοχή προστίθονται στο τέλος του μετώπου



Μέτωπο = [S0]

Μέτωπο πριν την ταξινόμηση = [S1, S2], Μέτωπο μετά την ταξινόμηση = [S1, S2]

Μέτωπο πριν την ταξινόμηση = [S2, S5, S4, S3], Μέτωπο μετά την ταξινόμηση = [S3, S4, S2, S5]

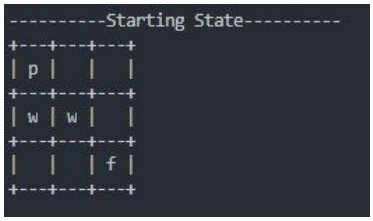
Μέτωπο πριν την ταξινόμηση = [S4, S2, S5, S7, S6], Μέτωπο μετά την ταξινόμηση = [S6, S7, S4, S2, S5]

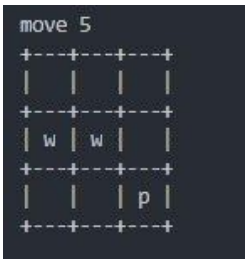
Μέτωπο πριν την ταξινόμηση = [S7, S4, S2, S5, S8, S9, S10], Μέτωπο μετά την ταξινόμηση = [S10, S8, S9, S7, S4, S2, S5]

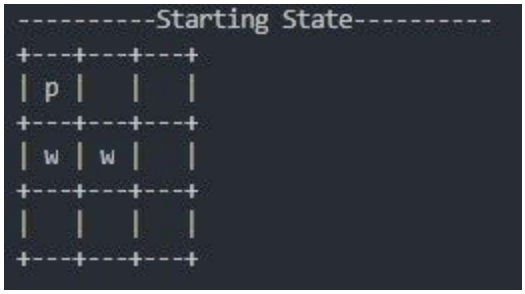
Τέλος γιατί στην κεφαλή του μετώπου βρίσκεται μια τελική κατάσταση.

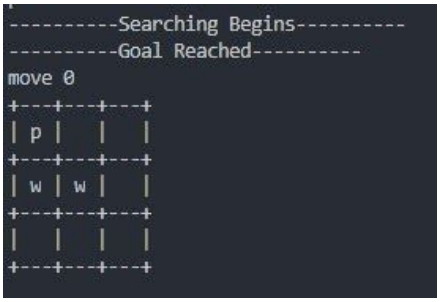
Τελικό μονοπάτι = [S0, S1, S3, S6, S10]

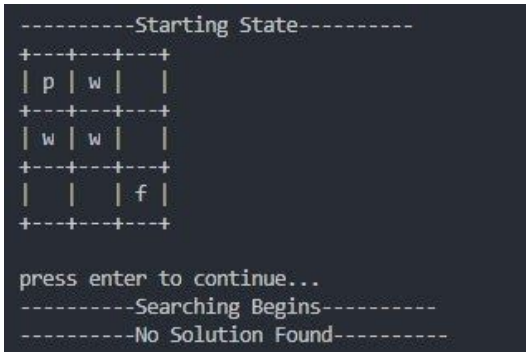
Ακραίες Καταστάσεις









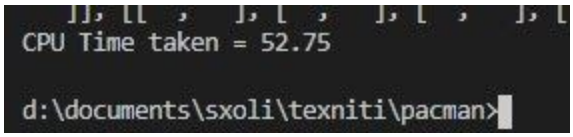


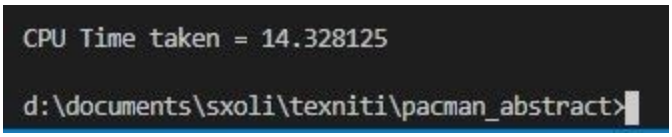
Αρχική κατάσταση χωρίς pacman δεν είναι εφικτή γιατί ο [constructor](#) τοποθετεί τον pacman σε τυχαία θέση χωρίς να εμπλέκεται ο χρήστης

Σημειώσεις

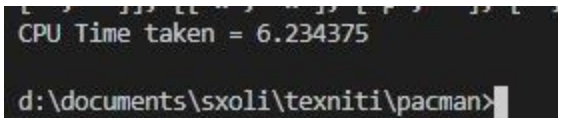
Η abstract προσέγγιση στο πρόβλημα κάνει την αναζήτηση εως και 3 φορές πιό γρήγορη. Οι παρακάτω εικόνες δείχνουν τους χρόνους που πήρε η ίδια αρχική κατάσταση στις δύο προσεγγίσεις.

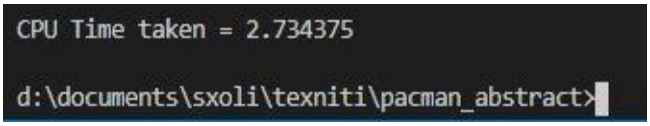
DFS

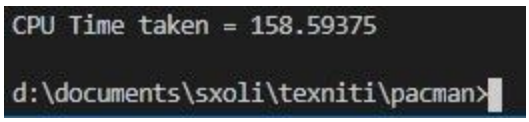


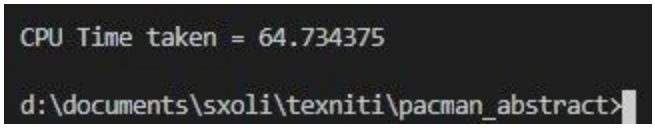


BFS









Ενδέχεται για μεγάλους κόσμους (5x5 με 10+ φρούτα) σε 32bit έκδοση της python το πρόγραμμα να πετάξει MemoryError. Αυτό γίνεται γιατί τα Windows (και τα περισσότερα OSes) περιορίζουν την μνήμη που δίνουν σε 32bit προγράμματα στα 2GB και η python ως δυναμική γλώσσα προγραμματισμού χρησιμοποιεί αρκετή μνήμη για κάθε object. Το πρόβλημα αυτό παρατηρήθηκε και στην κανονική έκδοση της εργασίας, όχι μόνο στην abstract. Το πρόβλημα λύνεται με εγκατάσταση 64bit έκδοσης της python.