ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ

Το πρόβλημα του parking

ΠΑΝΤΕΛΕΙΜΩΝ ΠΡΩΙΟΣ
ice18390023
5ο Εξάμηνο
ice18390023@uniwa.gr

ΔΕΣΠΟΙΝΑ ΛΥΚΟΥΔΗ
ice18390151
5ο Εξάμηνο
ice18390151@uniwa.gr

Τμήμα 1 (Νέοι) Δευτέρα 17:00-19:00

ΠΑΝΕΠΙΣΤΙΜΙΟ ΔΥΤΤΙΚΗΣ ΑΤΤΙΚΗΣ
UNIVERSITY OF WEST ATTICA

Υπεύθυνοι καθηγητές

ΓΕΩΡΓΟΥΛΗ ΑΙΚΑΤΕΡΙΝΗ
ΒΟΥΛΟΔΗΜΟΣ ΑΘΑΝΑΣΙΟΣ
ΚΟΛΛΙΑ ΗΛΙΑΝΝΑ
ΤΣΕΛΕΝΤΗ ΠΑΝΑΓΙΩΤΑ

Τμήμα Μηχανικών και Πληροφορικής Υπολογιστών

2020-2021

Περιεχόμενα

1 Ο κόσμος του προβλήματος

Ένας χώρος στάθμευσης (parking), διαθέτει ένα επίπεδο λειτουργίας με n χώρους στάθμευσης (spaces), αριθμημένους από S_1 έως S_n, όπου n ο αριθμός των χώρων. Ο χώρος εισόδου, είναι ο Space 1. Επίσης, υπάρχουν $n-1$ πλατφόρμες εναπόθεσης αυτοκινήτων (platforms), αριθμημένες από P_1 έως P_{n-1} και είναι τοποθετημένες ανά μια σε $n-1$ χώρους στάθμευσης. Στην αρχή λειτουργίας του προβλήματος, το parking είναι άδειο από αυτοκίνητα $(\forall \quad P_i = empty, \quad i \in \mathbb{N} \leq n-1)$, και στον χώρο υποδοχής δεν υπάρχει πλατφόρμα εναπόθεσης $(S_1 = empty)$ όπως φαίνεται στο σχήμα 2.1β΄ για $n = 4$. Τέλος, στην αρχή του προβλήματος, υπάρχουν 3 αυτοκίνητα σε αναμονή εισόδου στο parking.

Το parking διαθέτει αυτόματο σύστημα πλοηγού για την ρύθμιση της διαδικασίας εισόδου. Για να μπορέσει ένα αυτοκίνητο να μπει στο parking, θα πρέπει ο αυτόματος πλοηγός να εξασφαλίσει ότι υπάρχει μια ελεύθερη πλατφόρμα στο χώρο υποδοχής, και να εναποθέσει ένα αυτοκίνητο που είναι σε αναμονή εισόδου πάνω στην πλατφόρμα αυτή. Κάθε πλατφόρμα μπορεί να φέρει το πολύ ένα αυτοκίνητο.

Ο πλοηγός μπορεί να μετακινήσει μια πλατφόρμα από το χώρο που βρίσκεται σε έναν γειτονικό χώρο αρκεί αυτός να είναι κενός. Για παράδειγμα, στην αρχική κατάσταση, στον χώρο εισόδου S_1 μπορεί να μεταφερθεί η πλατφόρμα P_1 και ο χώρος S_2 πάνω στον οποίο βρισκόταν αυτή να γίνει κενός, ή να μεταφερθεί η πλατφόρμα P_3 με μεταβολή του αντίστοιχου χώρου S_4 σε κενό, αλλά η πλατφόρμα P_2 δεν μπορεί να μεταφερθεί. Κάθε πλατφόρμα που μεταφέρεται μπορεί να είναι είτε ελεύθερη είτε όχι (δηλαδή κατειλημμένη από αυτοκίνητο).

1.1 Προσδιορισμός του κόσμου

Ο κόσμος του προβλήματος προσδιορίζεται από:

- αντικείμενα
    1. χώροι (spaces)
    2. πλατφόρμες (platforms)
    3. αυτοκίνητα (cars)
- σχέσεις
    1. ο χώρος (space) 1 είναι η είσοδος των αυτοκινήτων
    2. κάθε χώρος φέρει μία πλατφόρμα εκτός από έναν που είναι άδειος
    3. κάθε πλατφόρμα μπορεί να έχει ένα αυτοκίνητο ή να είναι άδεια
- ιδιότητές
    1. ένας χώρος θεωρείται ελεύθερος αν δεν φέρει πλατφόρμα και έτσι μία πλατφόρμα μπορεί να μετακινηθεί σε αυτόν οριζόντια ή κάθετα αρκεί να είναι γείτονας.
    2. μία πλατφόρμα θεωρείται ελεύθερη αν δεν φέρει αυτοκίνητο και έτσι ένα αυτοκίνητο μπορεί να εισέλθει σε μία πλατφόρμα αρκεί να είναι άδεια και να βρίσκεται στον χώρο της εισόδου (space 1 στην προκειμένη περίπτωση)

1.2 Αρχική κατάσταση

Η αρχική κατάσταση του προβλήματος αναπαριστάτε στο σχήμα 2.1β΄ όπου ο χώρος 1 είναι η είσοδος των αυτοκινήτων και δεν φέρει πλατφόρμα, ενώ όλοι οι υπόλοιποι χώροι φέρουν και είναι διαθέσιμη για κάποιο αυτοκίνητο.

1.3 Τελική κατάσταση

Η τελική κατάσταση έχει πολλές μορφές. Στην τελική κατάσταση οι πλατφόρμες θα είναι τοποθετημένες οπουδήποτε στους χώρους και θα ισχύει μια από τις δύο περιπτώσεις:

1. ο αριθμός των αυτοκινήτων αναμονής είναι μεγαλύτερος από το 0 και όλες οι πλατφόρμες θα είναι μη διαθέσιμες, δηλαδή
   αυτοκίνητα αναμονής > 0 ∪ όλες οι μη διαθέσιμες πλατφόρμες = 0.

2. δεν θα υπάρχουν αυτοκίνητα αναμονής και οι μη διαθέσιμες πλατφόρμες θα είναι σίγουρα μικρότερες ή ίσες από το γινόμενο των στηλών και γραμμών μείον 1. Επίσης, θα πρέπει να είναι μεγαλύτερος ή ίσος με το μηδέν οι μη διαθέσιμες πλατφόρμες, δηλαδή
   αυτοκίνητα αναμονής = 0 ∪((στήλες × γραμμές) -1 ≥ μη διαθέσιμες πλατφόρμες ≥ 0).

Επίσης παρατηρούμε ότι, πάντα ο άδειος χώρος όταν βρεθεί η τελική κατάσταση θα είναι σε κάποιον γείτονα της εισόδου (space 1). Αυτό συμβαίνει επειδή, χρειάζεται να είναι άδειος ο χώρος εισόδου και μετά να μεταφερθεί η άδεια πλατφόρμα επάνω στον άδειο χώρο αφήνοντας άδειο τον χώρο στον οποίο βρισκόταν, ο οποίος είναι γείτονας τους χώρου εισόδου. Με αυτό τον τρόπο, περιορίζονται κατά ελάχιστα οι διαφορετικοί συνδυασμοί με τον περιορισμό πως ο κενός χώρος θα είναι πάντα γείτονας του χώρου εισόδου. Όμως, αυτό ισχύει μόνο για ένα πρόγραμμα, το οποίο δεν θα παραλείψει από σφάλμα κάποια τελική κατάσταση, αλλά και πάλι δεν παύουν και οι υπόλοιπες περιπτώσεις, όπου ο άδειος χώρος δεν είναι γείτονας του χώρου εισόδου να είναι τελικές καταστάσεις.

2 Αλλαγές αρχικού κώδικα και αποτελέσματα μετώπου

Σε αυτήν την ενότητα γίνονται αλλαγές στην απαρίθμηση των χώρων και στους τελεστές μετάβασης βάσει αλγορίθμου και όχι πίνακα.

2.1 Διαφοροποίηση του τελεστή μετάβασης καταστάσεων γειτόνων

Ο αλγόριθμος DFS με παρακολούθηση μετώπου, που δόθηκε από το εργαστήριο έχει τροποποιηθεί κατάλληλα για να υποστηρίζει διαφορετικά μεγέθη parking. Στην αρχή, τροποποιήσαμε την συνάρτηση ***neighbour*** προσθέτοντας της μια ακόμα παράμετρο, η οποία θα ήταν 0 ή 1 αναλόγως τον γείτονα που θα θέλαμε βάση τη λίστα spaces που απεικονίζεται στο σχήμα (2.1β΄), έτσι ώστε να απλοποιήσουμε τις δύο παρόμοιες συναρτήσεις ***neighbour1*** και ***neighbour2***. Ωστόσο, δεν ήταν τόσο ικανοποιητικός ο περιορισμός της λίστας ***spaces***. Για αυτόν τον λόγο, η λίστα spaces αντικαταστάθηκε από τις ακέραιες μεταβλητές ***n*** και ***m***, των οποίων το γινόμενο τους $m \times n$, δηλαδή οι γραμμές $\times$ τις στήλες, είναι το μέγεθος του πίνακα. Πλέον, με αυτήν την υλοποίηση δεν περιοριζόμαστε στην υπάρξη της λίστας spaces, που είναι περιοριστική για μεγάλα και διαφορετικά προβλήματα. Οπότε πλέον, η συνάρτηση ***neighbours*** του κώδικα 3.1 δέχεται ως δεύτερη παράμετρο τις λέξεις:

- UP
- DOWN
- LEFT
- RIGHT

Πλέων έχουμε 5 τελεστές μετάβασης από μία κατάσταση σε μία άλλη. Η συνάρτηση ***neighbours(state, neighbour)*** βρίσκει τον χώρο ο οποίος είναι άδειος στην κατάσταση που είναι η πρώτη παράμετρος και στην συνέχεια βάση της δεύτερης παραμέτρου βρίσκει τον γείτονα της εάν έχει. Η συνάρτηση επιστρέφει την νέα κατάσταση ή None εάν δεν είναι έγκυρη. Έστω ότι ο γείτονας είναι:

- UP, τότε θα πρέπει να ισχύει $i + n \leq m \cdot n$ για να έχει πάνω γείτονα
- DOWN, τότε θα πρέπει να ισχύει $i - n > 0$ για να έχει κάτω γείτονα
- RIGHT, τότε θα πρέπει να ισχύει i mod n $\neq 0$ για να έχει δεξιά γείτονα
- LEFT, τότε θα πρέπει να ισχύει i mod n $\neq 1$ για να έχει αριστερά γείτονα

Στο σχήμα 2.2 φαίνεται πως διαμορφώνεται ο τυφλός αλγόριθμος BFS και η ουρά λύσης μέχρι την τελική κατάσταση, με αρχική κατάσταση του σχήματος 2.1α΄. Ο BFS αν και παρέχει την μικρότερη ουρά επεκτείνει κάθε γείτονα με αποτέλεσμα να είναι αργός και χρονοβόρος μέχρι να φτάσει στην τελική κατάσταση.

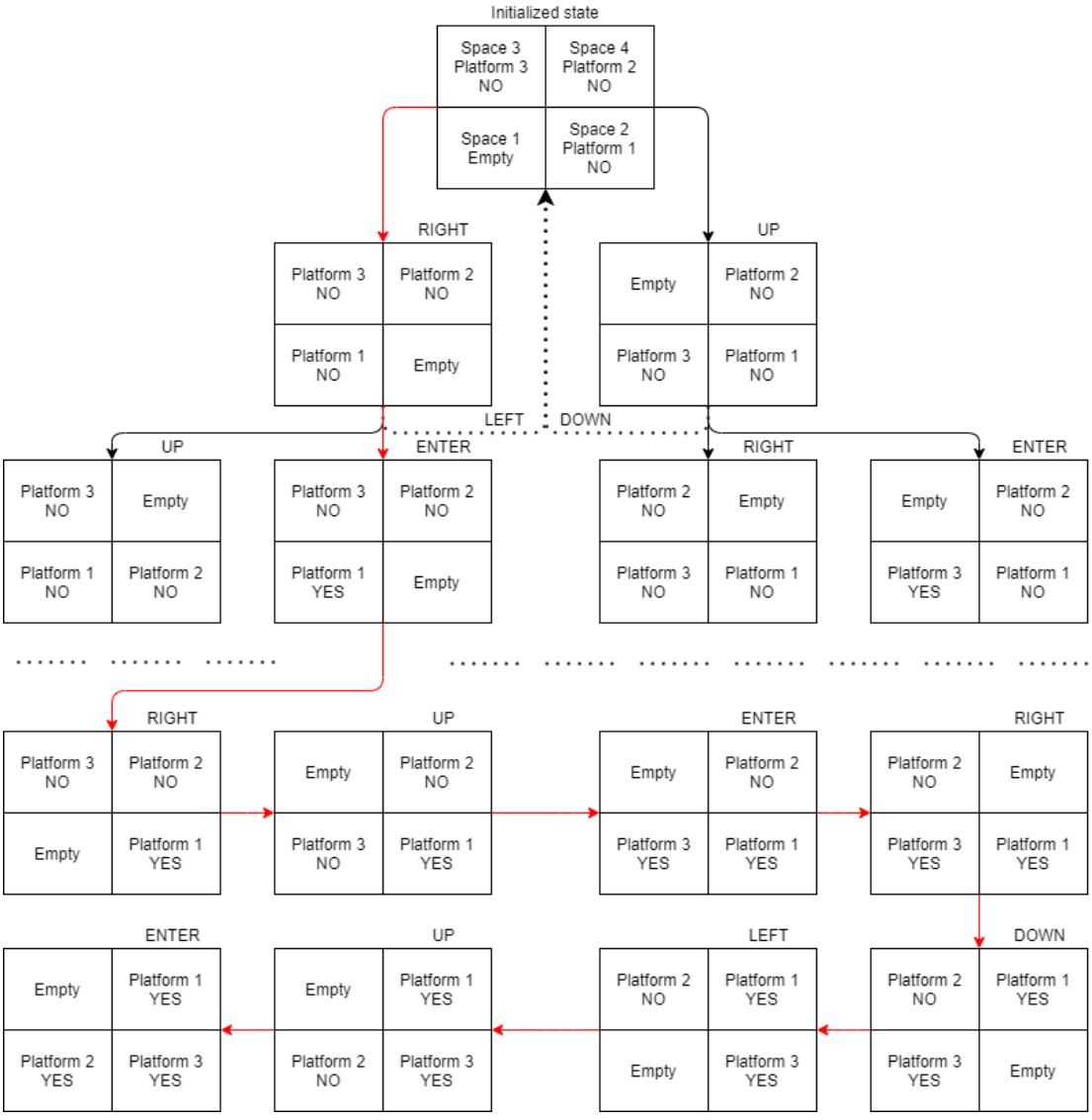
Space 3 Platform 3 NO	Space 4 Platform 2 NO
Space 1 Empty	Space 2 Platform 1 NO

(α′) Αρχική κατάσταση σε διαστάσεις 2×2 με διαφορετική σειρά απαρίθμησης των χώρων.

Space 4 Platform 3 NO	Space 3 Platform 2 NO
Space 1 Empty	Space 2 Platform 1 NO

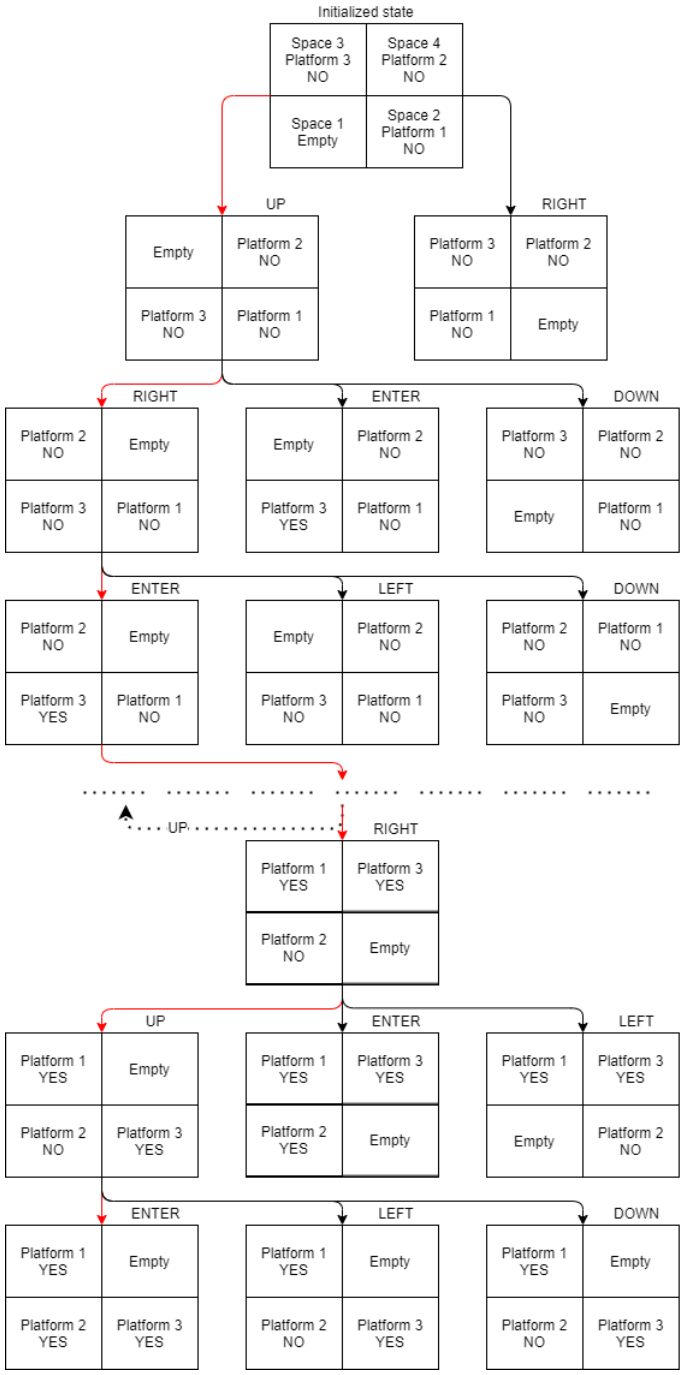
(β′) Αρχική κατάσταση σε διαστάσεις 2×2 με απαρίθμηση των χώρων όπως δόθηκε από το εργαστήριο.

Σχήμα 2.1: Αλλαγή απαρίθμησης των χώρων.



Σχήμα 2.2: BFS 2×2 χώρου

Στο σχήμα 2.3 φαίνεται πως επεκτείνεται ο τυφλός αλγόριθμος DFS μέχρι να βρει τελική κατάσταση, για την αρχική κατάσταση του σχήματος 2.1α΄. Ο DFS δίνει γρηγορότερη από τον BFS αλλά όχι αποτελεσματική.



Σχήμα 2.3: DFS 2×2 χώρου

2.2 Το μέτωπο με μέθοδο αναζήτησης DFS

Ο κώδικας 3.1 για την μέθοδο αναζήτησης DFS, με την αρχική κατάσταση του σχήματος (2.1α΄), βρίσκει τελική κατάσταση μετά από 18 επαναλήψεις της find_solution και το μέτωπο πριν βρει τελική κατάσταση είναι διαμορφωμένο όπως στον πίνακα 2.1. Παρατηρούμε ότι υπάρχει στο μέτωπο ήδη τελική κατάσταση, η οποία είναι με κόκκινο χρώμα και αυτό οφείλεται στην σειρά που τοποθετούνται τα αποτελέσματα των τελεστών στο μέτωπο. Επίσης, υπάρχει μια κατάσταση η οποία είναι πολύ καλύτερη σε σύγκρισή με τις προηγούμενες της και ενώ έχει βρεθεί πριν από πολλές επαναλήψεις βρίσκεται στην μέση του μετώπου παρόλο που χρειάζεται 5 στάδια μέχρι την τελική κατάσταση. Στον πίνακα 2.2 είναι το μέτωπο του DFS όταν βρίσκει την τελική κατάσταση.

Πίνακας 2.1: Μέτωπο του DFS πριν την τελική κατάσταση

Αναμονή αυτοκινήτων	space 1		space 2		space 3		space 4	
1	P2	NO	P3	YES	P1	YES	E	NO
0	P2	YES	E	NO	P1	YES	P3	YES
1	E	NO	P2	NO	P1	YES	P3	YES
1	P1	YES	E	NO	P3	YES	P2	NO
2	P1	NO	P2	NO	E	NO	P3	YES
2	P1	NO	E	NO	P3	YES	P2	NO
1	P1	YES	E	NO	P3	YES	P2	NO
2	E	NO	P1	NO	P3	YES	P2	NO
2	P3	YES	E	NO	P2	NO	P1	NO
3	P3	NO	P1	NO	E	NO	P2	NO
3	P3	NO	E	NO	P2	NO	P1	NO
2	P3	YES	P1	NO	E	NO	P2	NO
3	E	NO	P1	NO	P3	NO	P2	NO
3	P1	NO	E	NO	P3	NO	P2	NO

Πίνακας 2.2: Μέτωπο του DFS όταν βρήκε τελική κατάσταση

Αναμονή αυτοκινήτων	space 1		space 2		space 3		space 4	
0	P2	YES	P3	YES	P1	YES	E	NO
1	P2	NO	P3	YES	E	NO	P1	YES
1	P2	NO	E	NO	P1	YES	P3	YES
0	P2	YES	E	NO	P1	YES	P3	YES
1	E	NO	P2	NO	P1	YES	P3	YES
1	P1	YES	E	NO	P3	YES	P2	NO
2	P1	NO	P2	NO	E	NO	P3	YES
2	P1	NO	E	NO	P3	YES	P2	NO
1	P1	YES	E	NO	P3	YES	P2	NO
2	E	NO	P1	NO	P3	YES	P2	NO
2	P3	YES	E	NO	P2	NO	P1	NO
3	P3	NO	P1	NO	E	NO	P2	NO
3	P3	NO	E	NO	P2	NO	P1	NO
2	P3	YES	P1	NO	E	NO	P2	NO
3	E	NO	P1	NO	P3	NO	P2	NO
3	P1	NO	E	NO	P3	NO	P2	NO

2.3 Το μέτωπο με μέθοδο αναζήτησης BFS

Ο κώδικας 3.1 για την μέθοδο αναζήτησης BFS, με αρχική κατάσταση το σχήμα (2.1α΄), βρίσκει τελική κατάσταση μετά από 142 επαναλήψεις της find_solution. Ο πίνακας 2.3 παρουσιάζει το μέτωπο του BFS όταν βρίσκεται η τελική κατάσταση. Παρατηρούμε πως, στο μέτωπο την στιγμή που βρέθηκε η τελική κατάσταση, υπάρχει ακόμη μία τελική κατάσταση η οποία είναι με κόκκινο χρώμα.

Πίνακας 2.3: Μέτωπο του BFS

Αναμονή αυτοκινήτων	space 1		space 2		space 3		space 4	
0	P2	YES	P3	YES	E	NO	P1	YES
1	P2	NO	P3	YES	P1	YES	E	NO
1	E	NO	P3	NO	P2	YES	P1	YES
1	P2	YES	P3	NO	P1	YES	E	NO
1	E	NO	P2	YES	P1	YES	P3	NO
1	P1	YES	P2	YES	P3	NO	E	NO
2	P1	NO	E	NO	P3	NO	P2	YES
2	P1	NO	P2	YES	E	NO	P3	NO
1	P1	YES	P2	YES	P3	NO	E	NO
1	E	NO	P2	NO	P1	YES	P3	YES
0	P2	YES	E	NO	P1	YES	P3	YES
1	P2	NO	P3	YES	P1	YES	E	NO
1	E	NO	P2	YES	P1	NO	P3	YES
1	P2	YES	P3	YES	P1	NO	E	NO
1	E	NO	P3	YES	P2	YES	P1	NO
1	P3	YES	P1	NO	P2	YES	E	NO
2	P3	NO	E	NO	P2	YES	P1	NO
2	P3	NO	P1	NO	E	NO	P2	YES
1	P3	YES	P1	NO	P2	YES	E	NO

3 Η υλοποίηση του πλοηγού parking με DFS και BFS

Όταν ο διερμηνευτής (interpreter) διαβάζει έναν πηγαίο κώδικα θέτει μερικές ειδικές μεταβλητές. Μία από αυτές είναι η __name__ η οποία περιέχει το όνομα __main__. Το πρόγραμμα ξεκινάει λίγο πριν το τέλος του κώδικα 3.1 στην συνθήκη και συνεχίζει καλώντας την συνάρτηση main. Αν δεν υπήρχε η συνθήκη if __name__=="__main__" και η κλήση της συνάρτησης main γινόταν χωρίς εσοχή, πάλι θα είχαμε το ίδιο αποτέλεσμα. Αν το αφήναμε με την εσοχή θα ήταν συνέχεια της προηγούμενης συνάρτησης.

3.1 Η συνάρτηση main

Η μεταβλητή initial_state περιέχει την αρχική κατάσταση του προβλήματος. Το πρώτο στοιχείο περιέχει τα αυτοκίνητα τα οποία περιμένουν στην είσοδο για να μπουν σε κάποια πλατφόρμα, τα οποία μπορεί να είναι είτε παραπάνω απο τις διαθέσιμες πλατφόρμες που υπάρχουν είτε λιγότερα από τις πλατφόρμες που είναι διαθέσιμες είτε να μην υπάρχει κάποιο. Τα υπόλοιπα στοιχεία της λίστας είναι υπό-λίστες.

Ο δείκτης κάθε υπό-λίστας ταυτίζεται με τον χώρο που βρίσκεται απαριθμώντας με αύξουσα σειρά από κάτω αριστερά προς τα πάνω δεξιά και στο τέλος κάθε γραμμής συνεχίζει η αρίθμηση από τα αριστερά προς τα δεξιά. Η πρώτη παράμετρος του προγράμματος είναι η μέθοδος (μέχρι στιγμής έχουν υλοποιηθεί δύο μέθοδοι ο DFS και ο BFS) π.χ. αν το τρέξουμε από το IDE spyder δίνουμε στην κονσόλα runfile('/path_name/file_name', args='DFS'). Στην μεταβλητή method περνάει το πρώτο όρισμα το οποίο δόθηκε κατά την εκτέλεση του αρχείου και είναι ο τρόπος της μεθόδου αναζήτησης που χρησιμοποιείται για αναζήτηση. Έπειτα, εκτυπώνεται το '____BEGIN__SEARCHING____' και πριν κληθεί η find_solution καλείτε η make_front για να κατασκευάσει το μέτωπο και στη συνέχεια καλείται η find_solution με παραμέτρους το μέτωπο, μια κενή λίστα γιατί δεν έχουμε κάποια κατάσταση στο κλειστό σύνολο και την μέθοδο.

3.2 Η συνάρτηση make_front

Η συνάρτηση make_front κατασκευάζει το μέτωπο κάποιας κατάστασης και το επιστρέφει. Στην προκειμένη περίπτωση χρησιμοποιείται μόνο κατά την κλήση της find_solution στην αρχή της main και απλά επιστρέφει την ίδια κατάσταση, αλλά βάσει διαφορετικών μεθόδων αναζήτησής θα μπορούσε να χρησιμοποιείται για την κατασκευή του μετώπου με την σωστή σειρά.

3.3 Η συνάρτηση find_solution

Η find_solution παίρνει ως παραμέτρους το μέτωπο, το κλειστό σύνολο και την μέθοδο αναζήτησης. Στην αρχή ελέγχει αν υπάρχει μέτωπο, αν δεν υπάρχει τότε εκτυπώνεται κατάλληλο μήνυμα. Συνεχίζει τον έλεγχο αν το πρώτο στοιχείο του μετώπου βρίσκεται ήδη στο κλειστό σύνολο, δηλαδή έχει ξανά περάσει από αυτήν την κατάσταση στο παρελθόν. Αν η συνθήκη είναι αληθής τότε αντιγράφουμε το μέτωπο εξολοκλήρου σε μια νέα μεταβλητή για να μην επηρεαστεί κατά λάθος κάτι που δεν θέλουμε, αφαιρούμε την πρώτη κατάσταση που ανήκει ήδη στο μέτωπο και ξανά καλούμε την find_solution. Στην περίπτωση που καμία από

τις προηγούμενες συνθήκες δεν είναι αληθής, ελέγχεται αν το πρώτο στοιχείο του μετώπου είναι κάποια τελική κατάσταση και τότε εκτυπώνεται κατάλληλο μήνυμα.

Τέλος, αν όλες οι άλλες συνθήκες έχουν αποτύχει προστίθεται το πρώτο στοιχείο του μετώπου στο τέλος του κλειστού συνόλου, αντιγράφεται εξ ολοκλήρου το μέτωπο σε μια νέα μεταβλητή, καλείται η συνάρτηση expand_front με ορίσματα το νέο μέτωπο που μόλις αντιγράφτηκε και την μέθοδο αναζήτησης, στην συνέχεια αντιγράφεται το κλειστό σύνολο και τέλος καλείται η find_solution με παραμέτρους το μέτωπο το οποίο επέστρεψε η expand_front, το κλειστό σύνολο και την μέθοδο αναζήτησης.

3.4 Η συνάρτηση found_goal

Η συνάρτηση found_goal παίρνει ως παράμετρο μία κατάσταση, η οποία ελέγχεται για το αν είναι τελική κατάσταση. Πιο συγκεκριμένα, ελέγχει αν υπάρχουν άλλα αυτοκίνητα, εάν δεν υπάρχουν τότε το πρώτο στοιχείο της λίστας θα είναι 0 και η συνάρτηση επιστρέφει True. Σε αντίθετη περίπτωση, εκχωρούμε στην μεταβλητή length το μέγεθος της λίστας και από το δεύτερο στοιχείο έως και το τελευταίο ελέγχεται κάθε πλατφόρμα αν χρησιμοποιείται από κάποιο αυτοκίνητο, εάν κάποια πλατφόρμα δεν χρησιμοποιείται από κάποιο αυτοκίνητο, τότε η συνάρτηση επιστρέφει False, αλλιώς βγαίνει από τον βρόγχο και επιστρέφει ξανά True, εφόσον δεν βρεθεί αχρησιμοποίητη πλατφόρμα.

3.5 Η συνάρτηση expand_front

Η συνάρτηση expand_front παίρνει ως παραμέτρους το μέτωπο και την μέθοδο αναζήτησης. Οι υλοποιήσιμοι μέθοδοι αναζήτησης είναι οι αλγόριθμοι αναζήτησης κατά βάθος (DFS) και αναζήτησης κατά πλάτος (BFS). Η συνθήκη ελέγχει πρώτα τον τρόπο αναζήτησης, αν ο τρόπος αναζήτησης είναι DFS, τότε η πρώτη συνθήκη θα είναι αληθής, ενώ αν είναι BFS, η δεύτερη συνθήκη θα είναι αληθής. Η πρώτη συνθήκη όπως και η δεύτερη αρχικά ελέγχουν αν είναι κενό το μέτωπο έπειτα αφού εκτυπώσουν το μήνυμα 'Front:' και το παρόν μέτωπο, βγάζουν το πρώτο στοιχείο του μετώπου για επεξεργασία. Ο βρόγχος επανάληψης για την DFS μέθοδο εκχωρεί στην αρχή της λίστας μετώπου τα στοιχεία της λίστας με την σειρά που επιστράφηκαν από την συνάρτηση find_children, ενώ η μέθοδος BFS εκχωρεί τα στοιχεία της λίστας με την σειρά που επιστράφηκαν από την συνάρτηση find_children στο τέλος της λίστας μετώπου. Τέλος, η συνάρτηση επιστρέφει την λίστα μετώπου.

3.6 Η συνάρτηση find_children

Η συνάρτηση find_children δέχεται ως όρισμα μια κατάσταση, για την οποία θα βρει τις έγκυρες παράγωγες καταστάσεις της βάσει των τελεστών μετάβασης. Αρχικά, η μεταβλητή children είναι κενή διότι τώρα θα βρεθούν τα παιδιά της. Αντιγράφουμε τα στοιχεία της κατάστασης στην μεταβλητή enter_state και δίνεται ως όρισμα κατά την κλήση της συνάρτησης enter. Έπειτα, ξανά αντιγράφεται εξολοκλήρου η δοθείσα κατάσταση σε τέσσερις διαφορετικές μεταβλητές και καλείται για κάθε μεταβλητή η συνάρτηση neighbours, με πρώτο όρισμα την κατάσταση και δεύτερο όρισμα μία γραμματοσειρά LEFT, RIGHT, UP και DOWN.

Οι δύο συναρτήσεις enter και neibghors επιστρέφουν μια κατάσταση ή None και εκχωρούνται οι επιστρεφόμενες τιμές στις κατάλληλες μεταβλητές. Οι συνθήκες ελέγχουν αν έχει

επιστραφεί η δεσμευμένη λέξη None. Εάν η μεταβλητή περιέχει None σημαίνει πως δεν υπήρχε νόμιμη/έγκυρη κατάσταση με τον συγκεκριμένο τελεστή μετάβασης. Αλλιώς εάν, η μεταβλητή δεν είναι None τότε εκχωρείται η κάθε μια στο τέλος της λίστας children με συγκεκριμένη σειρά. Η σειρά με την οποία πρέπει να εκχωρείται είναι λεξικογραφικά (προς αύξοντα αριθμό). Για αυτό, πρώτα είναι η κατάσταση που επέστρεψε η συνάρτηση με δεύτερο όρισμα 'DOWN', μετά η 'LEFT', ακολουθεί η enter_child, η 'Right' και τέλος η 'UP'.

3.7 Η συνάρτηση enter

Η συνάρτηση enter δέχεται ως όρισμα μια κατάσταση και ελέγχει εάν υπάρχουν αυτοκίνητα, εάν ο χώρος 1 (Space 1) έχει πλατφόρμα και εάν είναι αχρησιμοποίητη η πλατφόρμα. Αν η συνθήκη είναι αληθής τότε, κάποιο αυτοκίνητο αναμονής εισέρχεται στην πλατφόρμα της εισόδου. Πιο συγκεκριμένα, με την state[0]!=0 ελέγχεται το πρώτο στοιχείο της λίστας, το οποίο είναι το μέγεθος των αυτοκινήτων σε αναμονή. Με την state[1][0][0]=='P' ελέγχουμε εάν στον χώρο εισόδου (με την πρώτη αγκύλη που περιέχει τον αριθμό 1), το πρώτο στοιχείο της υπό-λίστας (με την δεύτερη αγκύλη που περιέχει τον αριθμό 0) περιλαμβάνει το γράμμα P (με την τρίτη αγκύλη που περιέχει τον αριθμό 0), το οποίο σηματοδοτεί πως υπάρχει πλατφόρμα ή δεν το περιέχει, όπου αυτο σημαίνει πως είναι ο κενός χώρος

Επιπρόσθετα, με την state[1][1]=='NO' γίνεται έλεγχος εάν η είσοδος (δηλαδή στον χώρο 1 που είναι πάντα το δεύτερο στοιχείο και έτσι η πρώτη αγκύλη είναι 1) είναι αχρησιμοποίητη (δηλαδή το δεύτερο στοιχείο της υπό-λίστας του δεύτερου στοιχείου είναι 'NO', έτσι η δεύτερη αγκύλη είναι επίσης 1). Εάν, η παραπάνω συνθήκη είναι αληθής, τότε το πρώτο στοιχείο της νέας κατάστασης είναι τα αυτοκίνητα που ήταν σε αναμονή εκτός ενός και το επιτυγχάνουμε με την state[0]-1, η οποία αφαιρεί 1 από το πρώτο στοιχείο της λίστας, δηλαδή εκεί που περιέχεται ο αριθμός τον αυτοκινήτων προς εξυπηρέτηση. Το δεύτερο στοιχείο της λίστας στην νέα κατάσταση είναι ο χώρος εισόδου όπου η πλατφόρμα του αλλάζει σε χρησιμοποιούμενη. Και τέλος τα υπόλοιπα παραμένουν ίδια με την state[2:].

3.8 Η συνάρτηση neighbours

Η συνάρτηση neighbours δέχεται ως πρώτο όρισμα μια κατάσταση και ως δεύτερο όρισμα μια γραμματοσειρά, που θα πρέπει να είναι είτε 'UP', είτε 'DOWN', είτε 'RIGHT', είτε 'LEFT'. Η συνάρτηση είναι ένας τελεστής μετάβασης, όπου θα προσπαθήσει να μεταβεί σε μια άλλη έγκυρη κατάσταση βάσει του δεύτερου ορίσματος.

Αρχικά, θα αναζητήσει στη λίστα state την υπό-λίστα που περιέχει τον χώρο που είναι κενός, γιατί μόνο σε αυτόν μπορεί να μεταφερθεί μια πλατφόρμα. Εάν δεν τον βρει, εκχωρεί στην μεταβλητή i την τιμή -1 αλλιώς τον δείκτη στον οποίο βρίσκεται. Ο πρώτος έλεγχος της συνθήκης είναι για να μεταφερθεί η πάνω πλατφόρμα κάτω με αποτέλεσμα να αδειάσει ο επάνω χώρος. Ελέγχει αν η μεταβλητή neighbour είναι η λέξη 'UP', αν το 'i' είναι διάφορο του -1, σε περίπτωση που δεν βρέθηκε άδειος χώρος και το άθροισμα του δείκτη 'i' συν 'n' (όπου 'n' το μέγεθος των στηλών του parking) είναι μικρότερο ή ίσο από το γινόμενο 'm' επί 'n' (δηλαδή του μεγέθους του parking) είναι αληθής τότε μεταφέρεται η πάνω πλατφόρμα στον κενό χώρο. Αν η neighbour περιέχει την μεταβλητή 'DOWN' τότε γίνεται έλεγχος αν η αφαίρεση 'i' μείον 'n' είναι θετικός αριθμός. Αν είναι τότε γίνεται η μεταφορά της κάτω πλατφόρμας και ο κάτω

χώρος μένει κενός. Επίσης, δεν χρειάζεται για την 'DOWN' να ελέγχει αν ο 'i' είναι -1, διότι ήδη ελέγχει αν το i-n είναι θετικός αριθμός. Αν η neighbour περιέχει την λέξη 'RIGHT' τότε γίνεται έλεγχος αν το 'i' είναι διάφορο του -1 και αν το i mod n είναι διάφορο του μηδενός. Για να έχει κάποιος χώρος γείτονα στα δεξιά πρέπει το υπόλοιπο του αριθμητή i με παρανομαστή το n να μην είναι 0 αλλιώς βρίσκετε στην τελευταία δεξιά στήλη.

Τέλος, αν η λέξη είναι 'LEFT' γίνεται έλεγχος του 'i' mod 'n' να είναι διάφορο του 1. Αυτό όμως, μπορεί να λυθεί εξ' αρχής στην πρώτη συνθήκη που αναζητείται ο κενός χώρος πολύ απλά αντί να εκχωρείται η τιμή -1 να γίνεται επιστροφή της τιμής None.

3.9 Η συνάρτηση swap

Η συνάρτηση swap δέχεται ως πρώτο όρισμα μία κατάσταση και ως δεύτερο και τρίτο όρισμα τους δείκτες που θα γίνει η αντιμετάθεση και επιστρέφει την νέα κατάσταση. Η python αντιμεταθέτει με εύκολο τρόπο σε σύγκριση με άλλες γλώσσες προγραμματισμού τα περιεχόμενα τον μεταβλητών ως εξής:
a = 5
b = 10
a, b = b, a
print(a)
10
print(b)
5

```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov  2 22:43:35 2020

@author: Padelis Proios, Despoina Lykoudi
"""

#   +-------+-------+
#   |   3   |   4   |
#   +-------+-------+
#   |   1   |   2   |
#   +-------+-------+
#       ^
#    entrance

import copy
import sys

sys.setrecursionlimit(10**6)

def enter(state):
    if state[0]!=0 and state[1][0][0]=='P' and state[1][1]=='NO':
        new_state=[state[0]-1] + [[state[1][0], 'YES']] + state[2:]
        return new_state


def swap(state_l, i, j):
    state_l[i], state_l[j] = state_l[j], state_l[i]
    return state_l

```

```
31
32 def neighbours(state,neighbour):
33
34     elem=['E','NO']
35     i=state.index(elem) if elem in state else -1
36     if neighbour=='UP' and i!=-1 and i+n<=(m*n):
37         swap(state, i, i+n)
38         return state
39
40     elif neighbour=='DOWN' and i-n>0:
41         swap(state, i, i-n)
42         return state
43
44     elif neighbour=='RIGHT' and i!=-1 and i%n!=0:
45         swap(state, i, i+1)
46         return state
47
48     elif neighbour=='LEFT' and i%n!=1:
49         swap(state, i, i-1)
50         return state
51
52
53 def find_children(state):
54
55     children=[]
56
57     enter_state=copy.deepcopy(state)
58     enter_child=enter(enter_state)
59
60     tr1_state=copy.deepcopy(state)
61     tr1_child=neighbours(tr1_state,'LEFT')
62
63     tr2_state=copy.deepcopy(state)
64     tr2_child=neighbours(tr2_state,'RIGHT')
65
66     tr3_state=copy.deepcopy(state)
67     tr3_child=neighbours(tr3_state,'UP')
68
69     tr4_state=copy.deepcopy(state)
70     tr4_child=neighbours(tr4_state,'DOWN')
71
72     if tr4_child is not None:
73         children.append(tr4_child)
74
75     if tr1_child is not None:
76         children.append(tr1_child)
77
78     if enter_child is not None:
79         children.append(enter_child)
80
81     if tr2_child is not None:
82         children.append(tr2_child)
83
84     if tr3_child is not None:
85         children.append(tr3_child)
86
87     return children
88
89
90 def make_front(state):
91     return [state]
92
```

```
93
94  def expand_front(front, method):
95
96      if method=='DFS':
97          if front:
98              print("Front:")
99              print(front)
100             node=front.pop(0)
101             for child in find_children(node):
102                 front.insert(0,child)
103
104     elif method=='BFS':
105         if front:
106             print("Front:")
107             print(front)
108             node=front.pop(0)
109             for child in find_children(node):
110                 front.append(child)
111     return front
112
113
114 def find_solution(front, closed, method):
115
116     if not front:
117         print('_NO_SOLUTION_FOUND_')
118
119     elif front[0] in closed:
120         new_front=copy.deepcopy(front)
121         new_front.pop(0)
122         find_solution(new_front, closed, method)
123
124     elif found_goal(front[0]):
125         print('_GOAL_FOUND_')
126         print(front[0])
127     else:
128         closed.append(front[0])
129         front_copy=copy.deepcopy(front)
130         front_children=expand_front(front_copy, method)
131         closed_copy=copy.deepcopy(closed)
132         find_solution(front_children, closed_copy, method)
133
134
135 def found_goal(state):
136
137     if state[0]==0: return True
138
139     length = len(state)
140     for i in range(1,length):
141         if state[i][1]=='NO' and state[i][0][0]=='P': return False
142
143     return True
144
145 #       space order
146 #    +-------+-------+
147 #    |   3   |   4   |
148 #    +-------+-------+
149 #    |   1   |   2   |
150 #    +-------+-------+
151 #        ^
152 #     entrance
153 #
154 #
```

```
155 #
156 #         init state
157 #    +-----------+-----------+
158 #    |    P3 NO  |    P2 NO  |
159 #    +-----------+-----------+
160 #    |    E  NO  |    P1 NO  |
161 #    +-----------+-----------+
162 #          ^
163 #       entrance
164 #
165 # Διαστάσης m*n = γραμμές επί στείλες
166 m=2
167 n=2
168
169 def main():
170
171     initial_state = [3, ['E', 'NO'], ['P1', 'NO'], ['P3', 'NO'], ['P2', 'NO']]
172
173     method = sys.argv[1]
174
175     print('____BEGIN__SEARCHING____')
176     find_solution(make_front(initial_state), [], method)
177
178
179 if __name__ == "__main__":
180     main()
```

Κώδικας 3.1: Υλοποίηση μεθόδου DFS και BFS με μέτωπο

4 Η υλοποίηση του πλοηγού parking με BestFS

Στον κώδικα 3.1 έχουν γίνει μερικές αλλαγές στις προϋπάρχουσες συναρτήσεις και έχουν προστεθεί κάποιες ακόμη. Επίσης, λίγο πριν το τέλος στην συνθήκη if __name__ == "__main__" έχουμε προσθέσει πέντε επιπλέον γραμμές, εκ των οποίων οι τέσσερις πρώτες βρίσκονται σε σχόλια και είναι για την ανακατεύθυνση των συναρτήσεων print σε προεπιλεγμένο αρχείο. Η τελευταία προσθήκη είναι μία εκτύπωση, η οποία θα εμφανίζεται στην κονσόλα.

4.1 Η συνάρτηση main

Πλέον η main εκτυπώνει το μήνυμα 'The initial state is:' και σε νέα γραμμή την αρχική κατάσταση με την συνάρτηση που έχουμε κατασκευάσει print_state. Επιπλέον, επειδή είχαμε απενεργοποιήσει την εκτύπωση από την συνάρτηση expand_front (παρόλα αυτά παραμένει σε σχόλια αλλά είναι προτιμότερα να γράφονται σε αρχείο για την καλύτερη ανάλυση), προσθέσαμε μια επιπλέον παράμετρο στο τέλος, έτσι ώστε να εκτυπώνει τον αριθμό επαναλήψεως της find_solution για να παίρνουμε μία στοιχειώδη ανταπόκριση από το πρόγραμμα και ότι δεν έχει κολλήσει.

Έχουμε ονοματίσει 2 λίστες η μία είναι με το όνομα P2P4P8 και είναι χώρος 3×3 με τις πλατφόρμες P2, P4 και P8 να είναι οι μόνες διαθέσιμες. Η δεύτερη ονομασία που έχουμε δώσει είναι επίσης για χώρο 3×3 αλλά όλες οι πλατφόρμες να είναι διαθέσιμες και το έχουμε ονομάσει 3by3AllNo.

4.2 Η συνάρτηση neighbours

Στο εξής, η συνάρτηση neighbours εκχωρεί στο i, None αν δεν βρεθεί χώρος που είναι άδειος και έτσι επιστρέφει εάν το i είναι None. Ακόμα, αφαιρέθηκε από τις συνθήκες 'UP' και 'RIGHT' ο επιπλέον έλεγχος $i \neq -1$ εφόσον πλέον δεν τίθεται τέτοιο ζήτημα.

4.3 Η συνάρτηση find_children

Στην συνάρτηση find_children θα μπορούσαμε να βάλουμε τον τελεστή enter να κάνει πρώτος append, γιατί είναι ένας από τους πιο χρήσιμους και έχει σημαντικό αντίκτυπο ειδικά στο DFS, αλλά και ένα μηδαμινό στο BFS και σε ισοβάθμιες του BestFS.

4.4 Η συνάρτηση expand_front

Στην συνάρτηση expand_front προσθέσαμε τον ευριστικό αλγόριθμο BestFS, για τον οποίο βρίσκει τα παιδιά του πρώτου μετώπου, τα τοποθετεί στο μέτωπο χωρίς να επηρεάζεται ο τρόπος ο οποίος τα τοποθετεί γιατί ταξινομούνται όλα βάση του βάρους τους από την μέθοδο λιστών .sort(), με κλειδί τη συνάρτηση cost που έχει υλοποιηθεί για τον υπολογισμό βάρους κάθε κατάστασης.

Αντικαταστήσαμε την print(front) με την συνάρτηση print_result, που κατασκευάσαμε για εκτύπωση αποτελεσμάτων. Η εκτύπωση αποτελεσμάτων είναι σε σχόλιο για λόγους απλοϊκότητας, όπως και οι προηγούμενοι τρόποι εκτύπωσης ενός μετώπου παραμένουν και αυτοί

σε σχόλια για την σύγκρισή της ομαλής λειτουργίας, σε σύγκριση με την υλοποιημένη συνάρτηση.

4.5 Η συνάρτηση find_solution

Στην συνάρτηση find_solution προσθέσαμε στο τέλος μια επιπλέον παράμετρο, η οποία αυξάνεται κατά 1 και εκτυπώνει τον αριθμό, δηλαδή μετράμε τις επανάληψεις της find_solution. Επίσης, για την συνθήκη που δεν υπάρχει front προσθέσαμε ένα μήνυμα εκτύπωσης, διότι πολλές φορές υπήρχε πρόβλημα με τις μεταβλητές m και n επειδή ήταν μικρότερες από το μέγεθος των χώρων. Τέλος, στην συνθήκη όπου το πρώτο στοιχείο του μετώπου είναι τελική κατάσταση, αντικαταστήσαμε την εκτύπωση μόνο του πρώτου στοιχείου του μετώπου με τέσσερις γραμμές κώδικα, όπου η πρώτη εκτυπώνει το κατάλληλο μήνυμα για τον τρόπο που θα αναπαρασταθεί το μέτωπο (δηλαδή από το goal state έως και το τελευταίο μέτωπο).

Η δεύτερη γραμμή καλεί την υλοποιημένη συνάρτηση αποτελεσμάτων και εμφανίζει το μέτωπο με πρώτο το goal state και στη συνέχεια τα υπόλοιπα και για αυτόν τον λόγο έχουμε προσθέτει τις δύο επόμενες γραμμές κώδικα, για να ξανά εμφανίσει στην κονσόλα ποία είναι η goal state για την οποία η συνθήκη ήταν αληθής, καθώς σε πολύ μεγάλα μέτωπα δεν έφτανε ο οπτικός χώρος της κονσόλας, εκτός και αν καταγράφονταν σε αρχείο. Επιπρόσθετα, υπάρχουν σε σχόλια ακόμα δύο γραμμές κώδικα, όπου η πρώτη αφαιρεί την goal state κατάσταση από το front και η δεύτερη ξανά καλεί την συνάρτηση find solution μέχρι να είναι αληθής η πρώτη συνθήκη όπου δεν θα υπάρχει μέτωπο και θα εκτυπώσει τα μηνύματα '_NO_SOLUTION_FOUND_' και 'Check n and m variables', αλλά στην πραγματικότητα δεν θα ισχύουν επειδή έχει γίνει εξαντλητικός έλεγχος.

4.6 Η νέα συνάρτηση print_state

Η συνάρτηση print_state δέχεται ως παράμετρο μία κατάσταση. Αρχικά, εκχωρεί στην μεταβλητή level το μέγεθος των γραμμών μείον 1, γιατί θέλουμε να ξεκινήσουμε από το τελευταίο επίπεδο τα εκτυπώνουμε και τέλος να έρθουμε στην αρχή. Μετά υπάρχει βρόγχος επανάληψης που επαναλαμβάνεται τόσες φορές όσες είναι και οι γραμμές. Εντός του βρόγχου υπάρχει ένας ακόμη βρόγχος, ο οποίος θα εκτελεστεί τόσες φορές όσες και οι στήλες, εκχωρώντας στην μεταβλητή index το γινόμενο του level (επιπέδου) και των στηλών προσθέτοντας τον δείκτη j, ο οποίος για n=3 θα παίρνει τις τιμές 1,2,3. Έπειτα, εκτυπώνει το πρώτο και δεύτερο στοιχείο του χώρου βάση του index, τα οποία είναι η πλατφόρμα αν δεν είναι ο κενός χώρος αλλιώς τον κενό χώρο και αν είναι διαθέσιμα, διαχωρίζοντας τα με δύο tab και ανάμεσα τον χαρακτήρα '-' και τερματίζοντας τον χώρο με επίσης δύο tab, αλλά ανάμεσα με τον χαρακτήρα '|'. Όταν τελειώσει η επανάληψη για τις στήλες με δείκτη j εκτυπώνει μια αλλαγή γραμμής και μειώνει το επίπεδο κατά 1.

Τέλος, όταν τερματίσει και η επανάληψη των γραμμών με δείκτη i εκτυπώνονται κατάλληλα τα αυτοκίνητα, τα οποία περιμένουν στην είσοδο και διαχωρίζεται η κατάσταση αυτή από τις ενδεχομένως επόμενες με τον χαρακτήρα '-' αναλόγως πόσες είναι η στήλες και τρεις αλλαγές γραμμής.

4.7 Η νέα συνάρτηση print_result

Η συνάρτηση print_result παίρνει ως πρώτη παράμετρο μία τριπλή λίστα, δηλαδή μία λίστα καταστάσεων, ως δεύτερη και τρίτη παράμετρο δέχεται δύο συμβολοσειρές οι οποίες θα είναι αυτές που θα εκτυπωθούν ανάμεσα από το τελικό αριθμό του αποτελέσματος, ο οποίος θα είναι είτε πόσες καταστάσεις υπάρχουν στο μέτωπο την στιγμή που βρέθηκε η τελική κατάσταση είτε πόσες καταστάσεις/τελεστές χρειάστηκαν αν είναι ουρά. Η συνάρτηση εκτυπώνει 3 αλλαγές γραμμής (υπάρχει και μία κρυφή της συνάρτησης εφόσον δεν το αλλάζουμε και το αφήνουμε default). Έπειτα, για κάθε στοιχείο της result καλεί την print_state η οποία υλοποιεί εκτύπωση μίας κατάστασης, οπότε δίνουμε ως όρισμα την κατάσταση που περιέχει η μεταβλητή i. Τέλος, εμφανίζει το μέγεθος της πρώτης παραμέτρου ανάμεσα από τις δοθέντες συμβολοσειρές.

4.8 Η νέα συνάρτηση cost

Η συνάρτηση cost υπολογίζει το βάρος μίας κατάστασης και το επιστρέφει. Δέχεται ως παράμετρο μια κατάσταση, αρχικοποιεί τις μεταβλητές weight, emptyY, emptyX, X και Y ίσον με 0 (πράγμα το οποίο δεν χρειάζεται για την python επειδή είναι garbage collector εκτός της weight γιατί αλλιώς θα βγάλει error). Υποθέτουμε πως κάθε χώρος του parking είναι καρτεσιανές συντεταγμένες (x,y), με τον χώρο εισόδου (space 1) να έχει τις συντεταγμένες (0,0) και του πιο βορειανατολικού χώρου να έχει τις συντεταγμένες (n-1,m-1).

Το βάρος κάθε κατάστασης υπολογίζεται βάση του Manhattan distance συγκρίνοντας την απόσταση για κάθε άδεια πλατφόρμα με τον χώρο εισόδου. Επιπροσθέτως, για κάθε διαθέσιμη πλατφόρμα συγκρίνουμε την απόσταση του άδειου χώρου από την πλατφόρμα και την είσοδο, προσπαθώντας να την περιορίσουμε και να μην είναι πίσω (δηλαδή βορειοανατολικά) από την διαθέσιμη πλατφόρμα αλλά μπροστά (δηλαδή νοτιοδυτικά). Πιο συγκεκριμένα, αφού γίνει η αχρείαστη αρχικοποίηση εκτός της weight, εκχωρούμε στην μεταβλητή elem τον τρόπο με τον οποίο αναπαριστούμε τον άδειο χώρο, τον αναζητούμε με την μέθοδο .index(elem) και εκχωρείται ο δείκτης του, που είναι και ο αριθμός του χώρου στον οποίο βρίσκεται. Αν δεν βρεθεί τέτοιος χώρος, τότε στο index εκχωρείται η δεσμευμένη λέξη None, μετά γίνεται έλεγχος αν ο index είναι None και αν είναι επιστρέφει None.

Αν δεν είναι None τότε συνεχίζει και έτσι βρίσκουμε το επίπεδο/ύψος/συντεταγμένες Y (level/height/coordinates Y) από την ακέραια διαίρεση του index-1 με το μέγεθος των γραμμών του πίνακα και την εκχωρούμε στην μεταβλητή emptyY. Αμέσως μετά, υπολογίζουμε την στήλη/πλάτος/συντεταγμένες X (column/width/coordinates X) από το υπόλοιπο της διαίρεσης του index-1 με το μέγεθος των στηλών και το εκχωρούμε στην μεταβλητή emptyX. Οπότε τώρα, η emptyX και emptyY περιέχουν τις συντεταγμένες του άδειου χώρου.

Για παράδειγμα, έστω ότι οι γραμμές m είναι 3 και ο στήλες n είναι 3. Ας υποθέσουμε επίσης ότι, ο άδειος χώρος που δεν φέρει καμία πλατφόρμα είναι ο χώρος 1 δηλαδή index = 1. Τότε, το αποτέλεσμα της ακέραιας διαίρεσης του (index-1) ÷ m, η οποία θα εκχωρηθεί στο emptyY, θα είναι 0 και το υπόλοιπο της, δηλαδή το αποτέλεσμα του (index-1) mod n, θα εκχωρηθεί στο emptyX και θα είναι επίσης 0. Έτσι, καταλήγουμε να έχουμε τις συντεταγμένες (emptyX,emptyY) = (0,0).

Έστω, για τις ίδιες διαστάσεις ο άδειος χώρος είναι ο 6, τότε η ακέραια διαίρεση του

(index-1) και m θα είναι emptyY = 1 και το υπόλοιπο της διαίρεσης (index-1) mod n είναι 2. Στην συνέχεια, υπάρχει ένας βρόγχος επανάληψης στου οποίου τις μεταβλητές εκχωρούνται τα αποτελέσματα της συνάρτησης που απαριθμεί κάθε κατάσταση, ξεκινώντας από το 1, η οποία απαρίθμηση αποσκοπεί στον αριθμό χώρου που βρίσκεται η πλατφόρμα. Πιο αναλυτικά, η συνάρτηση enumerate παράγει μία λίστα απαριθμώντας κάθε στοιχείο από την λίστα state που της δώσαμε από το δεύτερο της στοιχείο ξεκινώντας την απαρίθμηση από τον αριθμό 1 και εκχωρούνται στις μεταβλητές space, platform και avalilability ο αριθμός του χώρου, το όνομα της πλατφόρμας ή 'E' αν είναι ο άδειος χώρος και η διαθεσιμότητα αντίστοιχα.

Στην συνέχεια, γίνεται έλεγχος εάν είναι πλατφόρμα κρινοντάς το μόνο τον πρώτο χαρακτήρα από την μεταβλητή platform αν είναι 'P' και εάν είναι διαθέσιμη ελέγχοντας αν η μεταβλητή availability είναι 'NO'. Έπειτα, όπως και προηγουμένως, με τον άδειο χώρο υπολογίζουμε τις συντεταγμένες της διαθέσιμης πλατφόρμας και τις εκχωρούμε στις μεταβλητές X και Y.

Ο υπολογισμός της κατάστασης για κάθε διαθέσιμη πλατφόρμα είναι η συνάθροιση τριών σταδίων. Στο πρώτο στάδιο αυξάνουμε την μεταβλητή weight, από τον υπολογισμό Manhattan distance με σημεία τον άδειο χώρο και την πλατφόρμα Md((emptyX,emptyY), (X,Y)). Στο δεύτερο στάδιο προσθέτουμε στην weight το αποτέλεσμα της Manhattan distance με σημεία τον άδειο χώρο και την είσοδο (space 1) Md((emptyX,emptyY), (0,0)). Στο τρίτο στάδιο προσθέτουμε στην weight το αποτέλεσμα της Manhattan distance της διαθέσιμης πλατφόρμας από την είσοδο Md((X,Y), (0,0)). Τέλος, η συνάρτηση cost θα επιστρέψει την μεταβλητή weight το οποίο είναι το βάρος το οποίο βρήκε για αυτήν την κατάσταση.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov  2 22:43:35 2020

@author: Padelis Proios, Despoina Lykoudi
"""

#   +-------+-------+
#   |   3   |   4   |
#   +-------+-------+
#   |   1   |   2   |
#   +-------+-------+
#       ^
#    entrance

import copy
import sys

sys.setrecursionlimit(10**6)

def enter(state):
    if state[0]!=0 and state[1][0][0]=='P' and state[1][1]=='NO':
        new_state=[state[0]-1] + [[state[1][0], 'YES']] + state[2:]
        return new_state


def swap(state_l, i, j):
    state_l[i], state_l[j] = state_l[j], state_l[i]
    return state_l


```

```
32 def neighbours(state,neighbour):
33
34     elem=['E','NO']
35     i=state.index(elem) if elem in state else None
36
37     if i is None: return i
38
39     if neighbour=='UP' and i+n<=(m*n):
40         swap(state, i, i+n)
41         return state
42
43     elif neighbour=='DOWN' and i-n>0:
44         swap(state, i, i-n)
45         return state
46
47     elif neighbour=='RIGHT' and i%n!=0:
48         swap(state, i, i+1)
49         return state
50
51     elif neighbour=='LEFT' and i%n!=1:
52         swap(state, i, i-1)
53         return state
54
55
56 def find_children(state):
57
58     children=[]
59
60     enter_state=copy.deepcopy(state)
61     enter_child=enter(enter_state)
62
63     tr1_state=copy.deepcopy(state)
64     tr1_child=neighbours(tr1_state,'LEFT')
65
66     tr2_state=copy.deepcopy(state)
67     tr2_child=neighbours(tr2_state,'RIGHT')
68
69     tr3_state=copy.deepcopy(state)
70     tr3_child=neighbours(tr3_state,'UP')
71
72     tr4_state=copy.deepcopy(state)
73     tr4_child=neighbours(tr4_state,'DOWN')
74
75
76
77     if tr4_child is not None:
78         children.append(tr4_child)
79
80     if tr1_child is not None:
81         children.append(tr1_child)
82
83     if enter_child is not None:
84         children.append(enter_child)
85
86     if tr2_child is not None:
87         children.append(tr2_child)
88
89     if tr3_child is not None:
90         children.append(tr3_child)
91
92
93     return children
```

```
94
95
96  def make_front(state):
97      return [state]
98
99
100 def expand_front(front, method):
101
102     if method=='DFS':
103         if front:
104             #print("Front:")
105             #print(front)
106             #print_result(front, '\n ========= front has', 'states =========')
107             node=front.pop(0)
108             for child in find_children(node):
109                 front.insert(0,child)
110
111     elif method=='BFS':
112         if front:
113             #print("Front:")
114             #print(front)
115             #print_result(front, '\n ========= front has', 'states =========')
116             node=front.pop(0)
117             for child in find_children(node):
118                 front.append(child)
119
120     elif method=='BestFS':
121         if front:
122             #print("Front:")
123             #print(front)
124             #print_result(front, '\n ========= front has', 'states =========')
125             node=front.pop(0)
126             for child in find_children(node):
127                 front.insert(0,child)
128             front.sort(key=cost)
129
130     return front
131
132
133
134 def find_solution(front, closed, method, i):
135
136     i+=1
137     print(i);
138
139     if not front:
140         print('_NO_SOLUTION_FOUND_')
141         print('Check n and m variables')
142     elif front[0] in closed:
143         new_front=copy.deepcopy(front)
144         new_front.pop(0)
145         find_solution(new_front, closed, method,i)
146     elif found_goal(front[0]):
147         print('_GOAL_FOUND_')
148         print('\n\n ========= Printing front from front[0] until the end when goal
    state took place =========\n\n')
149         print_result(front, '\n ========= front has', 'states =========')
150         print('\n\nAnd the goal state of front[0] is:\n')
151         print_state(front[0])
152         #front.pop()
153         #find_solution(front, closed, method,i)
154
```

```
155     else:
156         closed.append(front[0])
157         front_copy=copy.deepcopy(front)
158         front_children=expand_front(front_copy, method)
159         closed_copy=copy.deepcopy(closed)
160         find_solution(front_children, closed_copy, method,i)
161 
162 
163 def found_goal(state):
164 
165     if state[0]==0: return True
166 
167     length = len(state)
168     for i in range(1,length):
169         if state[i][1]=='NO' and state[i][0][0]=='P': return False
170 
171     return True
172 
173 
174 def print_result( result, str1, str2):
175 
176     print('\n\n')
177 
178     for i in result:
179         print_state(i)
180 
181     print(str1, len(result), str2)
182 
183 
184 
185 def print_state(state):
186     level = m-1
187 
188     for i in range(0,m):
189 
190         for j in range(1,n+1):
191 
192             index = (level*n)+j
193             print( state[index][0], '\t-\t', state[index][1], '\t|\t', end = '')
194 
195         print()
196         level -= 1
197 
198     print('   ^^^\nCars waiting:', state[0])
199     print('-----------------', end = '')
200     for i in range(1,n):
201         print('--------------------',end='')
202     print('\n\n')
203 
204 
205 
206 def cost(state):
207     weight = 0
208     emptyY = 0
209     emptyX = 0
210     X = 0
211     Y = 0
212 
213     elem = ['E','NO']
214     index = state.index(elem) if elem in state else None
215 
216 
```

```
    if index is None: return index

    emptyY = (index-1)//m
    emptyX = (index-1)%n



    for space,[platform,availability] in enumerate(state[1:], start = 1):

        if platform[0] == 'P' and availability == 'NO':

            Y = (space-1)//m
            X = (space-1)%n

            weight += abs(Y-emptyY) + abs(X-emptyX)

            weight += emptyX + emptyY

            weight += Y + X

    return weight


#       space order
#   +-------+-------+
#   |   3   |   4   |
#   +-------+-------+
#   |   1   |   2   |
#   +-------+-------+
#       ^
#    entrance
#
#
#
#       init state
#   +-----------+-----------+
#   |   P3 NO   |   P2 NO   |
#   +-----------+-----------+
#   |   E  NO   |   P1 NO   |
#   +-----------+-----------+
#         ^
#     entrance
#
# Διαστάσης m*n = γραμμές επί στείλες

m=2
n=2

def main():

    # 2 x 2
    initial_state = [3, ['E', 'NO'], ['P1', 'NO'], ['P3', 'NO'], ['P2', 'NO']]
    #initial_state = [3, ['E', 'NO'], ['P1', 'NO'], ['P2', 'NO'], ['P3', 'NO']]
    #initial_state = [3, ['E', 'NO'], ['P1', 'YES'], ['P2', 'NO'], ['P3', 'YES']]
    #initial_state = [3, ['E', 'NO'], ['P1', 'YES'], ['P2', 'YES'], ['P3', 'NO']]
    #initial_state = [3, ['E', 'NO'], ['P1', 'NO'], ['P2', 'YES'], ['P3', 'NO']]
```

```
    # 3 x 2
  '''
    initial_state = [5, ['E', 'NO'],
                        ['P1', 'NO'],
                        ['P2', 'NO'],
                        ['P3', 'NO'],
                        ['P4', 'NO'],
                        ['P5', 'NO']]
  '''

    # 3 x 3 All No
  '''
    initial_state = [8, ['E', 'NO'],
                        ['P1', 'NO'],
                        ['P2', 'NO'],
                        ['P3', 'NO'],
                        ['P4', 'NO'],
                        ['P5', 'NO'],
                        ['P6', 'NO'],
                        ['P7', 'NO'],
                        ['P8', 'NO']]
  '''


    method = sys.argv[1]

    print('The initial state is:')
    print_state(initial_state)

    print('____BEGIN__SEARCHING____')
    find_solution( make_front(initial_state), [], method, 0)


if __name__ == "__main__":
    #stdoutold = sys.stdout
    #sys.stdout = fd = open('/path/to/output1.txt', 'w')
    main()
    #sys.stdout = stdoutold
    #fd.close()
    print('\n\n========== Done ==========')
```

Κώδικας 4.1: Προσθήκη BestFS και αλλαγές του κώδικα 3.1.

5 Υλοποίηση ουράς

Ο κώδικας 4.1 τροποποιήθηκε έτσι ώστε να υποστηρίζει παρακολούθηση ουράς.

5.1 Αντικατάσταση της make_front με την make_queue

Η make_queue κάνει ακριβώς ότι και η make_front απλώς αλλάξαμε το όνομα για λόγους κατανόησης και συμβατότητας.

5.2 Αντικατάσταση της extend_front με την extend_queue

Αρχικά, άλλαξαν τα ονόματα και όπου είχαμε front πλέον έχουμε queue. Επίσης, με αυτήν την αλλαγή τώρα δεν έχουμε μόνο το μέτωπο αλλά όλο το μονοπάτι που έχει διανύσει έως τώρα το κάθε μέτωπο από την ρίζα (αρχική κατάσταση). Το μέτωπο δεν παρέχεται άμεσα αλλά έμμεσα, είναι το τελευταίο στοιχείο κάθε ουράς. Η μόνη αλλαγή είναι το κλειδί για τον BestFS που δεν παύει να είναι το ίδιο με την διαφορά πως υποστηρίζει πλέον ουρά και όχι μόνο ένα state. Έτσι, πράττουμε όπως και πριν με αυτήν την διαφοροποίηση.

5.3 Τροποποίηση της συνάρτησης find_solution

Τώρα, η δεύτερη συνθήκη ελέγχει εάν το τελευταίο στοιχείο της πρώτης λίστας της μεταβλητής queue υπάρχει στο κλειστό σύνολο, διότι πλέον το μέτωπο αντιπροσωπεύεται από το τελευταίο στοιχείο κάθε στοιχείου της queue. Επίσης, έχουμε αντικαταστήσει την τρίτη συνθήκη, αφού ελέγξει αν το τελευταίο στοιχείο της πρώτης λίστας είναι σε τελική κατάσταση, εκτυπώνεται κατάλληλο μήνυμα και έπειτα όλο το μέτωπο και το μέγεθος του μετώπου. Στη συνέχεια, εκτυπώνεται κατάλληλο μήνυμα, καθώς η ουρά από την ρίζα μέχρι την τελική κατάσταση.

5.4 Η νέα συνάρτηση cost_queue

Η cost_queue δέχεται ως παράμετρο μία ουρά, καλεί την συνάρτηση cost με όρισμα το τελευταίο στοιχείο της ουράς (δηλαδή για αυτό που θέλουμε να υπολογίσουμε το βάρος και βρίσκεται στο μέτωπο) και επιστρέφει την τιμή που της επέστρεψε η συνάρτηση cost.

5.5 Τροποποίηση της συνάρτησης cost

Στην συνάρτηση cost βρήκαμε έναν ακόμη καλύτερο τρόπο υπολογισμού του βάρους, αλλά προτιμήσαμε να κρατήσουμε τον original. Ωστόσο, υπάρχει σε σχόλια και αυτό που κάνει είναι να πολλαπλασιάζει το πλήθος τον διαθέσιμων πλατφορμών με την απόσταση της κάθε διαθέσιμης πλατφόρμας και ήταν αποτελεσματικό. Πιο αναλυτικά, συγκρίναμε τον αρχικό με τον τροποποιημένο, τον οποίο αποκαλούμε πολλαπλασιασμό διαθέσιμων πλατφορμών. Στον πίνακα 5.1 παρατηρούμε πως για αρχική κατάσταση 2×2 και όλες οι πλατφόρμες διαθέσιμες (2by2AllNo) όπως στο σχήμα 2.1α΄ δεν υπάρχει καμία απολύτως διαφορά. Για αρχική κατάσταση 3×3 επίσης με όλες τις πλατφόρμες διαθέσιμες, (3by3AllNo) υπάρχει μια μικρή διαφορά για την ουρά. Αντιθέτως, στην αρχική κατάσταση 3×3, με μόνο 3

Πίνακας 5.1: Συγκρίσεις αρχικού cost και πολλαπλασιασμένου από τις διαθέσιμες θέσεις.

2by2AllNo	Αρχικός	Πολλαπλασιασμός διαθέσιμων πλατφορμών
find_solution	11	11
front	14	14
queue	11	11
3by3AllNo		
find_solution	181	74
front	129	107
queue	62	58
P2P4P8		
find_solution	63	32
front	72	51
queue	43	27
P2P8P10P11		
find_solution	127	89
front	116	105
queue	64	58
3by4AllNo		
find_solution		154
front		206
queue		103

πλατφόρμες διαθέσιμες, (P2P4P8) η ουρά του αρχικού ήταν κατά πολύ μεγαλύτερη. Επίσης, παραλείποντας την αρχική κατάσταση για πίνακά 3×4 με τέσσερις διαθέσιμες πλατφόρμες (P2P8P10P11), εστιάζουμε για την αρχική κατάσταση πίνακα 3×4 με όλες τις πλατφόρμες διαθέσιμες (3by4AllNo), για τον οποίο δεν έχουμε πληροφορίες για τον αρχικό τρόπο επειδή δεν έφτασε σε αποτέλεσμα ποτέ, σε αντίθεση με την τροποποίηση του. Ακόμα, υπάρχουν αρχεία με τα κατάλληλα ονόματα και τα αποτελέσματα των στοιχείων του πίνακα 5.1 ή μπορεί κάποιος να παράξει τα αρχεία μόνος του με τον κώδικα 5.1.

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Mon Nov  2 22:43:35 2020
4
5  @author: Padelis Proios, Despoina Lykoudi
6  """
7
8  #   +-------+-------+
9  #   |   3   |   4   |
10 #   +-------+-------+
11 #   |   1   |   2   |
12 #   +-------+-------+
13 #       ^
14 #    entrance
15
```

```
16 import copy
17 import sys
18
19 sys.setrecursionlimit(10**6)
20
21 def enter(state):
22     if state[0]!=0 and state[1][0][0]=='P' and state[1][1]=='NO':
23         new_state=[state[0]-1] + [[state[1][0], 'YES']] + state[2:]
24         return new_state
25
26
27
28 def swap(state_l, i, j):
29     state_l[i], state_l[j] = state_l[j], state_l[i]
30     return state_l
31
32
33
34 def neighbours(state,neighbour):
35
36     elem=['E','NO']
37     i=state.index(elem) if elem in state else None
38
39     if i is None: return i
40
41     if neighbour=='UP' and i+n<=(m*n):
42         swap(state, i, i+n)
43         return state
44
45     elif neighbour=='DOWN' and i-n>0:
46         swap(state, i, i-n)
47         return state
48
49     elif neighbour=='RIGHT' and i%n!=0:
50         swap(state, i, i+1)
51         return state
52
53     elif neighbour=='LEFT' and i%n!=1:
54         swap(state, i, i-1)
55         return state
56
57
58
59 def find_children(state):
60
61     children=[]
62
63     enter_state=copy.deepcopy(state)
64     enter_child=enter(enter_state)
65
66     tr1_state=copy.deepcopy(state)
67     tr1_child=neighbours(tr1_state,'LEFT')
68
69     tr2_state=copy.deepcopy(state)
70     tr2_child=neighbours(tr2_state,'RIGHT')
71
72     tr3_state=copy.deepcopy(state)
73     tr3_child=neighbours(tr3_state,'UP')
74
75     tr4_state=copy.deepcopy(state)
76     tr4_child=neighbours(tr4_state,'DOWN')
77
```

```
78
79
80      if tr4_child is not None:
81          children.append(tr4_child)
82
83      if tr1_child is not None:
84          children.append(tr1_child)
85
86      if enter_child is not None:
87          children.append(enter_child)
88
89      if tr2_child is not None:
90          children.append(tr2_child)
91
92      if tr3_child is not None:
93          children.append(tr3_child)
94
95
96      return children
97
98
99
100 def extend_queue(queue, method):
101     if method=='DFS':
102         # Εκτήπωση όλων των ουρών
103         #for i in range(0, len(queue)):
104         #    print("Queue",i,':')
105         #    print_result(queue[i],'=== queue has','states ===\n\n')
106
107         # Εκτήπωση του μετώπου
108         #for q in queue:
109         #    print_state(q[-1])
110         #print('\n ========= front has', len(queue) ,'states =========\n\n')
111         node=queue.pop(0)
112         queue_copy=copy.deepcopy(queue)
113         children=find_children(node[-1])
114         for child in children:
115             path=copy.deepcopy(node)
116             path.append(child)
117             queue_copy.insert(0,path)
118
119     elif method=='BFS':
120         # Εκτήπωση όλων των ουρών
121         #for i in range(0, len(queue)):
122         #    print("Queue",i,':')
123         #    print_result(queue[i],'=== queue has','states ===\n\n')
124
125         # Εκτήπωση του μετώπου
126         #for q in queue:
127         #    print_state(q[-1])
128         #print('\n ========= front has', len(queue) ,'states =========\n\n')
129         node=queue.pop(0)
130         queue_copy=copy.deepcopy(queue)
131         children=find_children(node[-1])
132         for child in children:
133             path=copy.deepcopy(node)
134             path.append(child)
135             queue_copy.append(path)
136
137
138
139
```

```
140     elif method=='BestFS':
141         # Εκτήπωση όλων των ουρών
142         #for i in range(0, len(queue)):
143         #     print("Queue",i,':')
144         #     print_result(queue[i],'=== queue has','states ===\n\n')
145 
146         # Εκτήπωση του μετώπου
147         #for q in queue:
148         #     print_state(q[-1])
149         #print('\n ========= front has', len(queue) ,'states =========\n\n')
150         node=queue.pop(0)
151         queue_copy=copy.deepcopy(queue)
152         children=find_children(node[-1])
153         for child in children:
154             path=copy.deepcopy(node)
155             path.append(child)
156             queue_copy.insert(0,path)
157         queue_copy.sort(key=cost_queue)
158 
159     return queue_copy
160 
161 
162 
163 def find_solution( queue, closed, method, i):
164 
165     i+=1
166     print(i);
167 
168 
169     if not queue:
170         print('_NO_SOLUTION_FOUND_')
171         print('Check n and m variables')
172 
173     elif queue[0][-1] in closed:
174         new_queue=copy.deepcopy(queue)
175         new_queue.pop(0)
176         find_solution( new_queue, closed, method,i)
177 
178     elif found_goal(queue[0][-1]):
179         print('_GOAL_FOUND_')
180 
181         print('\n\n ======== Printing front from front[0] until the end ========\n\n')
182         for q in queue:
183             print_state(q[-1])
184         print('\n ========= front has', len(queue) ,'states =========')
185 
186         print('\n\n ======== Printing queue from root until the goal state ========\n')
187 
188         print_result( queue[0], '\n ======== we need', 'steps until success ========')
189 
190 
191     else:
192         closed.append(queue[0][-1])
193         queue_copy=copy.deepcopy(queue)
194         queue_children=extend_queue(queue_copy, method)
195         closed_copy=copy.deepcopy(closed)
196         find_solution(queue_children, closed_copy, method,i)
197 
198 
199 
200 
201 
```

```
202 def found_goal(state):
203 
204     if state[0]==0: return True
205 
206     length = len(state)
207     for i in range(1,length):
208         if state[i][1]=='NO' and state[i][0][0]=='P': return False
209 
210     return True
211 
212 
213 
214 def print_result( result, str1, str2):
215 
216     print('\n\n')
217 
218     for i in result:
219         print_state(i)
220 
221     print(str1, len(result), str2)
222 
223 
224 
225 def print_state(state):
226     level = m-1
227 
228     for i in range(1,m+1):
229 
230         for j in range(1,n+1):
231 
232             index = (level*n)+j
233             print( state[index][0], '\t-\t', state[index][1], '\t|\t', end = '')
234 
235         print()
236         level -= 1
237 
238     print('   ^^^\nCars waiting:', state[0])
239     print('-----------------', end = '')
240     for i in range(1,n):
241         print('--------------------',end='')
242     print('\n\n')
243 
244 
245 
246 def cost_queue(queue):
247     return cost(queue[-1])
248 
249 
250 
251 def cost(state):
252     weight = 0
253     emptyY = 0
254     emptyX = 0
255     X = 0
256     Y = 0
257 
258     elem = ['E','NO']
259     index = state.index(elem) if elem in state else None
260 
261     if index is None: return index
262 
263 
```

```
264     emptyY = (index-1)//m
265     emptyX = (index-1)%n
266 
267     #freePlats = 0
268     #for platform,availability in state[1:]:
269     #    if platform[0] == 'P' and availability == 'NO':
270     #        freePlats +=1
271 
272     for space,[platform,availability] in enumerate(state[1:], start = 1):
273 
274         if platform[0] == 'P' and availability == 'NO':
275 
276             Y = (space-1)//m
277             X = (space-1)%n
278 
279             weight += abs(X-emptyX) + abs(Y-emptyY)
280 
281             weight += (emptyX + emptyY)
282 
283             # multiply by avalilable platfroms (( Y + X + 1 ) * freePlats)
284             weight += ( X + Y  )  #* freePlats
285 
286     return weight
287 
288 
289 
290 #        space order
291 #    +-------+-------+
292 #    |   3   |   4   |
293 #    +-------+-------+
294 #    |   1   |   2   |
295 #    +-------+-------+
296 #        ^
297 #     entrance
298 #
299 #
300 #
301 #        init state
302 #    +-----------+-----------+
303 #    |   P3 NO   |   P2 NO   |
304 #    +-----------+-----------+
305 #    |   E  NO   |   P1 NO   |
306 #    +-----------+-----------+
307 #          ^
308 #      entrance
309 #
310 # Διαστάσης m*n = γραμμές επί στείλες
311 
312 m=2
313 n=2
314 
315 def main():
316 
317     # 2 x 2
318     initial_state = [3, ['E', 'NO'], ['P1', 'NO'], ['P3', 'NO'], ['P2', 'NO']]
319     #initial_state = [3, ['E', 'NO'], ['P1', 'NO'], ['P2', 'NO'], ['P3', 'NO']]
320     #initial_state = [3, ['E', 'NO'], ['P1', 'YES'], ['P2', 'NO'], ['P3', 'YES']]
321     #initial_state = [3, ['E', 'NO'], ['P1', 'YES'], ['P2', 'YES'], ['P3', 'NO']]
322     #initial_state = [3, ['E', 'NO'], ['P1', 'NO'], ['P2', 'YES'], ['P3', 'NO']]
323 
324 
325 
```

```
    # 3 x 2
    '''
    initial_state = [5, ['E', 'NO'],
                        ['P1', 'NO'],
                        ['P2', 'NO'],
                        ['P3', 'NO'],
                        ['P4', 'NO'],
                        ['P5', 'NO']]
    '''

    # 3 x 3 All No
    '''
    initial_state = [8, ['E', 'NO'],
                        ['P1', 'NO'],
                        ['P2', 'NO'],
                        ['P3', 'NO'],
                        ['P4', 'NO'],
                        ['P5', 'NO'],
                        ['P6', 'NO'],
                        ['P7', 'NO'],
                        ['P8', 'NO']]
    '''




    method = sys.argv[1]

    print('The initial state is:')
    print_state(initial_state)

    print('____BEGIN__SEARCHING____')
    find_solution( make_queue(initial_state), [], method, 0)



if __name__ == "__main__":
    #stdoutold = sys.stdout
    #sys.stdout = fd = open('C:\\full\\path\\to\\output.txt', 'w')
    main()
    #sys.stdout = stdoutold
    #fd.close()
    print('\n\n========== Done ==========')
```

Κώδικας 5.1: Υλοποίηση με παρακολούθηση ουράς.

Αναφορές

[1] Νικόλαος Αβούρης, Μιχαήλ Κουκιάς, Βασίλης Παλιουράς, και Κυριάκος Σγάρμπας. Python. ***ΕΙΣΑΓΩΓΗ ΣΤΟΥΣ ΥΠΟΛΟΓΙΣΤΕΣ***, 4η, 2018. https://www.cup.gr/book/isagong-stous-ypologistes-me-ti-glossa-python/.

[2] Αικατερίνη Γεωργούλη. Τεχνητή νοημοσύνη. ***ΘΕΩΡΗΤΙΚΗ ΠΡΟΣΕΓΓΙΣΗ ΤΗΣ ΕΠΙΛΥΣΗΣ ΠΡΟΒΛΗΜΑΤΩΝ ΜΕ ΤΗ ΒΟΗΘΕΙΑ ΜΕΘΟΔΩΝ ΤΕΧΝΗΤΗΣ ΝΟΗΜΟΣΥΝΗΣ***, 2015. http://hdl.handle.net/11419/3381.

[3] Amit Patel. Heuristics. 2018. http://theory.stanford.edu/ amitp/GameProgramming/.