Σταυρουλάκη Μαρία, 3160168
Τσισκάκης Φώτιος, 3170162

1η ΕΡΓΑΣΙΑ ΤΕΧΝΗΤΗΣ ΝΟΗΜΟΣΥΝΗΣ

Διάσχιση της γέφυρας

1.ΤΡΟΠΟΣ ΧΡΗΣΗΣ

Η εργασία που αποφασίσαμε να εκπονήσουμε είναι η διάσχιση γέφυρας. Η χρήση του προγράμματος είναι ιδιαίτερα απλή. Από το χρήστη ζητείται να εισάγει τον αριθμό των ατόμων που θέλει να περάσουν στην απέναντι όχθη, τα ονόματα τους καθώς και ο χρόνος που απαιτείται από τον καθένα ώστε να διασχίσει τη γέφυρα.

2.ΛΕΙΤΟΥΡΓΙΑ ΠΡΟΓΡΑΜΜΑΤΟΣ ΚΑΙ ΑΛΓΟΡΙΘΜΟΣ ΤΕΧΝΗΤΗΣ ΝΟΗΜΟΣΥΝΗΣ

Κατά την εκτέλεση του προγράμματος, και αφού το πρόγραμμα έχει λάβει τις τιμές από το χρήστη, τοποθετεί όλα τα άτομα στη δεξιά μεριά και φτιάχνει μία αρχική κατάσταση. Έπειτα εκτελείται ο αλγόριθμος A*, ο οποίος λειτουργεί με τον εξής τρόπο: τοποθετεί στο χώρο αναζήτησης ολόκληρη τη δεξιά μεριά, που είναι η αρχική κατάσταση και για όσο έχουμε καταστάσεις στον χώρο αναζήτησης εκτελεί τον παρακάτω βρόγχο. Αφαιρεί από τον χώρο αναζήτησης την πρώτη κατάσταση, που έχει το μικρότερο κόστος(κάνοντας sort στον χώρο αναζήτησης με βάση το f(s)), και ελέγχει αν είναι τελική κατάσταση. Στην περίπτωση που είναι τότε η λύση βρέθηκε. Διαφορετικά παράγονται όλα τα παιδιά της (αν το torch είναι στα δεξιά παράγονται μόνο τα παιδιά που περνάνε έναν μόνο άνθρωπο από αριστερά στα δεξιά και όμοια από δεξιά στα αριστερά περνάνε μόνο 2 άτομα για να μειώσουμε κατά πολύ τον αριθμό των παιδιών) ,δηλαδή όλες τις πιθανές καταστάσεις που μπορούν να προκύψουν από την αρχική, και τις ταξινομεί με βάση το ολικό κόστος, το οποίο είναι η ευρετική συνάρτηση συν το κόστος από την ρίζα μέχρι την τωρινή κατάσταση. Άμα δεν μείνει καμία κατάσταση στον χώρο αναζήτησης και δεν έχει βρεθεί λύση, τότε το πρόγραμμα ενημερώνει τον χρήστη κατάλληλα και τερματίζει. Άμα βρεθεί λύση εμφανίζεται στην οθόνη το μονοπάτι που ακολούθησε ο αλγόριθμος.

Για την κατασκευή της ευρετικής αφαιρέσαμε τον περιορισμούς του μέγιστου αριθμού ατόμων που μπορούν να περάσουν τη γέφυρα. Άμα η λάμπα είναι στα αριστερά τότε το κόστος της ευρετική αποτελείται από τον ελάχιστο χρόνο διάσχισης ενός ατόμου που βρίσκεται στα αριστερά συν το μέγιστο χρόνο διάσχισης ενός ατόμου που βρίσκεται στα δεξιά. Όμοια άμα η λάμπα είναι στα δεξιά τότε η ευρετική επιστρέφει το μεγαλύτερο κόστος

ανθρώπου που βρίσκεται στα δεξιά. Η παραπάνω ευρετική βγήκε από την αφαίρεση περιορισμών του αρχικού προβλήματος οπότε είναι αποδεκτή, ο αριθμός των παιδιών της κάθε κατάστασης είναι πεπερασμένος και το κόστος μετάβασης είναι θετικό άρα ξέρουμε από την θεωρία ότι ο A* είναι πλήρης και βέλτιστος.

3.ΠΕΡΙΓΡΑΦΗ ΚΛΑΣΕΩΝ

Έχουμε υλοποιήσει τις εξής κλάσεις : Person,State,SpaceSearcher και Main.

Η κλάση Person αντιπροσωπεύει τον κάθε άνθρωπο και περιέχει το όνομά του και τον χρόνο που κάνει να διασχίσει μόνος του την γέφυρα.

Η κλάση State αντιπροσωπεύει μία κατάσταση του προβλήματος και έχει σαν γνωρίσματα τα εξής:

- starting_time : Μία μεταβλητή για το χρόνο που πρέπει να έχει ολοκληρωθεί το πρόβλημα που είναι και static γιατί δεν αλλάζει από κατάσταση σε κατάσταση
- left_side, right_side : Δύο μεταβλητές arraylist που περιέχουν τα αντικείμενα των ανθρώπων στην δεξιά και στην αριστερή μεριά της γέφυρας αντίστοιχα
- father : Μία μεταβλητή State που περιέχει μία αναφορά στον πατέρα της κατάστασης
- score : Μία μεταβλητή int που περιέχει την τιμή της ευρετικής συνάρτησης
- total_cost : Μία μεταβλητή int που περιέχει το συνολικό κόστος από την ρίζα μέχρι την τωρινή κατάσταση συν την τιμή της ευρετικής
- right_side_bool : Μία μεταβλητή boolean που μας δείχνει που είναι η λάμπα κάθε φορά(true = δεξιά, false = αριστερά)
- time left : Μία μεταβλητή int που περιέχει πόσο χρόνος μένει από την αρχική κατάσταση που ξεκινήσαμε μέχρι την τωρινή

Η κλάση State περιέχει τις εξής συναρτήσεις :

- 2 constructors : ένας για την αρχική κατάσταση και ένας για τα παιδιά που ουσιαστικά αντιγράφουμε την αρχική κατάσταση και αλλάζουμε θέση στην λάμπα(την αλλαγή των ανθρώπων την κάνουμε σε επόμενο βήμα)
- Setters και Getters για left_side, right_side, time_left
- isRight_side_bool() : επιστρέφει τη μεταβλητή right_side_bool

- getCostFromRoot() : επιστρέφει το κόστος από την ρίζα μέχρι την τωρινή κατάσταση
- getTotalCost() : επιστρέφει το κόστος από την ρίζα μέχρι την τωρινή κατάσταση συν την τιμή της ευρετικής
- move_left(Person[] peopleToMove) : μετακινεί όλους τους ανθρώπου που βρίσκονται στο array που παίρνει σαν είσοδο από τα δεξιά στα αριστερά(η λάμπα έχει μετακινηθεί όταν φτιάχνουμε την κατάσταση) και υπολογίζει το κόστος της κίνησης
- move_right(Person[] peopleToMove) : μετακινεί όλους τους ανθρώπου που βρίσκονται στο array που παίρνει σαν είσοδο από τα αριστερά στα δεξιά(η λάμπα έχει μετακινηθεί όταν φτιάχνουμε την κατάσταση) και υπολογίζει το κόστος της κίνησης
- isTerminal() : επιστρέφει αν είναι τελική η κατάσταση ελέγχωντας άμα η λίστα με τα άτομα που είναι δεξιά είναι άδεια(άρα όλα τα άτομα θα έχουν πάει αριστερά)
- print() : εκτυπώνει την τωρινή κατάσταση, για παράδειγμα: A B -------- 💡 C D E
- getChildren() : παράγει όλα τα παιδιά της τωρινής κατάστασης και τα επιστρέφει σε μία arraylist
- heuristic() : υπολογίζει την τιμή της ευρετικής και την επιστρέφει
- cartesianProduct(ArrayList<Person> list) : παράγει το καρτεσιανό γινόμενο της λίστας που παίρνει σαν είσοδο και το επιστρέφει με μορφή ενός arraylist με πίνακες δύο θέσεων που περιέχει είτε 2 διαφορετικά άτομα που θέλουμε να περάσουν στην απέναντι μεριά ή 2 ίδια άτομα όταν θ΄πελουμε να περάσει μόνο ένα άτομο. Ειδικότερα έχουμε τη μορφή [1,2] όταν θέλουμε να περάσει ο 1 και 2 ενώ [2,2] για να περάσει μόνο ο 2. Σε αυτό το σημείο έχουμε προσθέσει σαν βελτίωση τον περιορισμό όταν η λάμπα είναι δεξιά να περνάνε σίγουρα 2 άτομα, διότι δε θα ήταν συμφέρον να περάσει μόνο ένας. Ομοίως, αν η λάμπα είναι στα δεξιά να περνάει μόνο ένας για τον ίδιο λόγο.
- getFather() : επιστρέφει την κατάσταση πατέρα της τωρινής κατάστασης, το χρησιμοποιούμε για να βρούμε το path της λύσης
- compareTo(Object o) : κάνουμε override το compareTo γιατί όταν ταξινομούμε τις καταστάσεις πρέπει αυτό να γίνεται με βάση το συνολικό τους κόστος (g(s)+h(s))

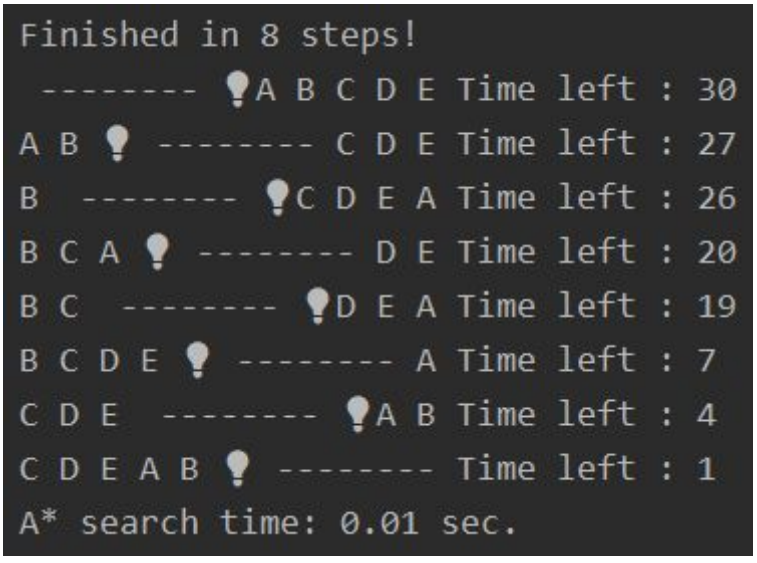
Η κλάση SpaceSearcher υλοποιεί τον αλγόριθμο A*. Περιέχει μόνο ένα γνώρισμα, το states, το οποίο αποτελεί τον χώρο αναζήτησης του προβλήματος που περιέχει αντικείμενα states. Επίσης περιέχει μία συνάρτηση

την Astar που παίρνει σαν είσοδο την αρχική κατάσταση και εκτελεί τον αλγόριθμο A*.

Η κλάση Main που αποτελεί την εκτελέσιμη κλάση. Περιέχει ένα arraylist με τα άτομα της αρχικής κατάστασης, την init που ζητάει από τον χρήστη τιμές για τον αριθμό των ατόμων, τα ονόματά τους ,το χρόνο που κάνει το καθένα να διασχίσει τη γέφυρα καθώς και τον μέγιστο χρόνο που θέλει ο χρηστης να έχουν περάσει όλη στην απέναντι μεριά. Η initTest κατασκευάζει το πρόβλημα που είχε στο βίντεο της εκφώνησης. Τέλος, η Main τρέχει μία από τις παραπάνω συναρτήσεις(στο τελικό την init) και στην συνέχεια φτιάχνει το αντικείμενο state για την αρχική κατάσταση και τρέχει τον αλγόριθμο από τον SpaceSearcher. Άμα ο αλγόριθμος βρει λύση εμφανίζει τα βήματα που χρειάστηκε και το μονοπάτι από την ρίζα μέχρι την λύση. Αλλιώς, αν δεν βρει λύση εμφανίζει το παρακάτω μήνυμα: “Could not find solution”. Επίσης η Main υπολογίζει το πόσο χρόνο χρειάστηκε ο αλγόριθμος και το εμφανίζει.

4.ΠΑΡΑΔΕΙΓΜΑΤΑ

Με βάση το παράδειγμα του βίντεο παίρνουμε το εξής αποτέλεσμα:



Χωρίς την βελτίωση για τον περιορισμό των ατόμων όταν παράγουμε τα παιδιά παίρνουμε το ίδιο αποτέλεσμα, αλλά ο χρόνος είναι πολύ μεγαλύτερος.

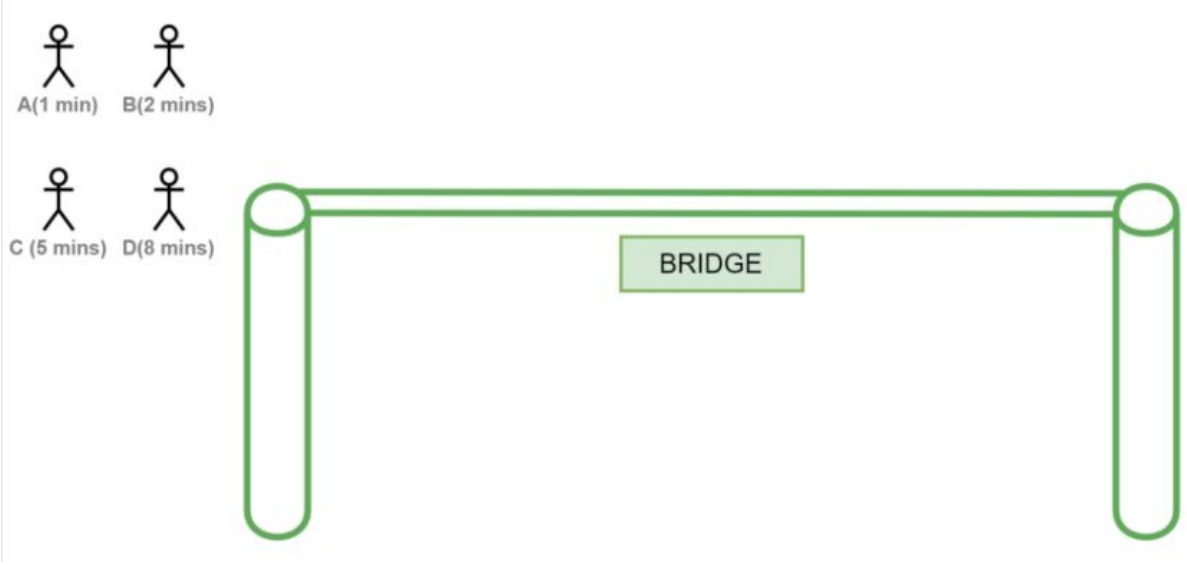
```
Finished in 8 steps!
 -------- A B C D E Time left : 30
A B  -------- C D E Time left : 27
B  -------- C D E A Time left : 26
B C A  -------- D E Time left : 20
B C  -------- D E A Time left : 19
B C D E  -------- A Time left : 7
C D E  -------- A B Time left : 4
C D E A B  -------- Time left : 1
A* search time: 1.026 sec.
```

Εκτελώντας το πρόβλημα με ίδιο αριθμό ατόμων και τιμών για το χρόνο τους, αλλά με μέγιστο χρόνο 25 παίρνουμε το παρακάτω αποτέλεσμα.

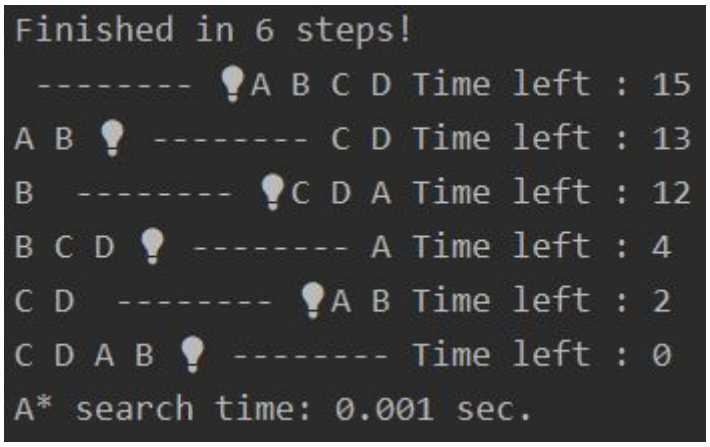
```
Could not find solution
A* search time: 0.022 sec.
```

Άρα μπορεί να βρεθεί λύση σε μέγιστο χρόνο τουλάχιστον 29.

Επίσης λύσαμε το εξής πρόβλημα:



με τη μόνη διαφορά ότι ξεκινούν στα δεξιά και καταλήγουν στα αριστερά. Ο μέγιστος χρόνος του προβλήματος είναι 15. Το πρόβλημα το βρήκαμε στο site https://www.geeksforgeeks.org/puzzle-18-torch-and-bridge/.



5.ΣΧΟΛΙΑ

Για την αναπαράσταση της λάμπας χρησιμοποιήσαμε το emoji με κωδικο *\uD83D\uDCA1*. Σε περίπτωση που δε σας το εμφανίζει βρίσκεται στις γραμμές 115 και 118 της κλάσης State.