ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

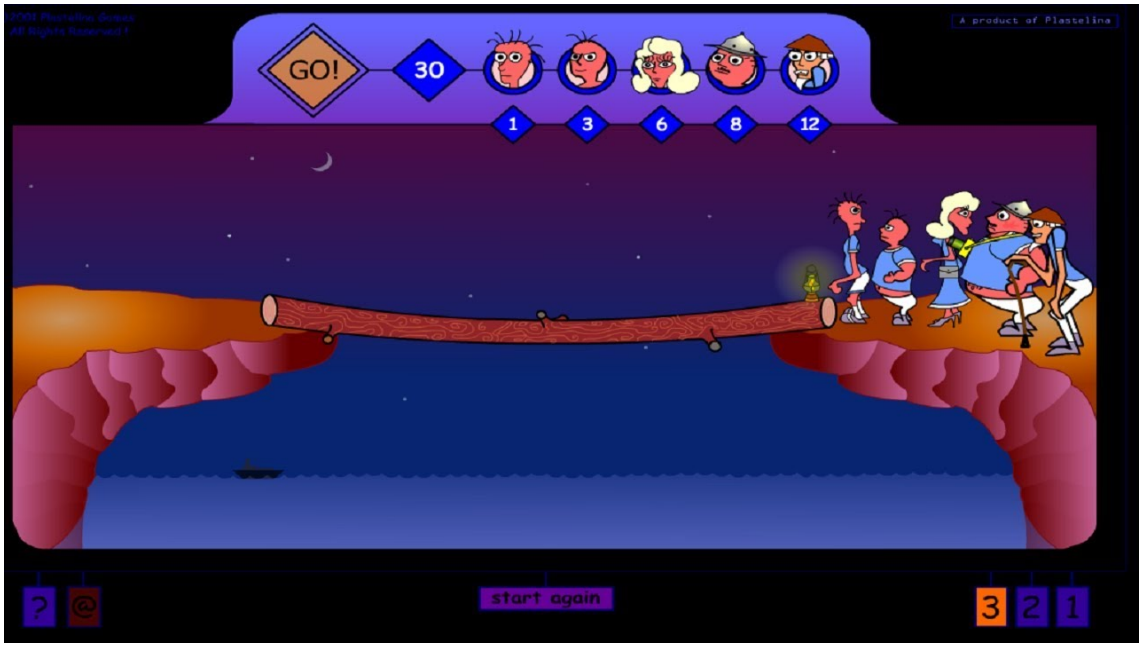
Τεχνητή Νοημοσύνη
1η Εργασία

Παναγιώτης Κάτσος(3180077)
Επαμεινώνδας Ιωάννου (3140059)
Πέτρος Τσότσι(3180193)

2020-2021

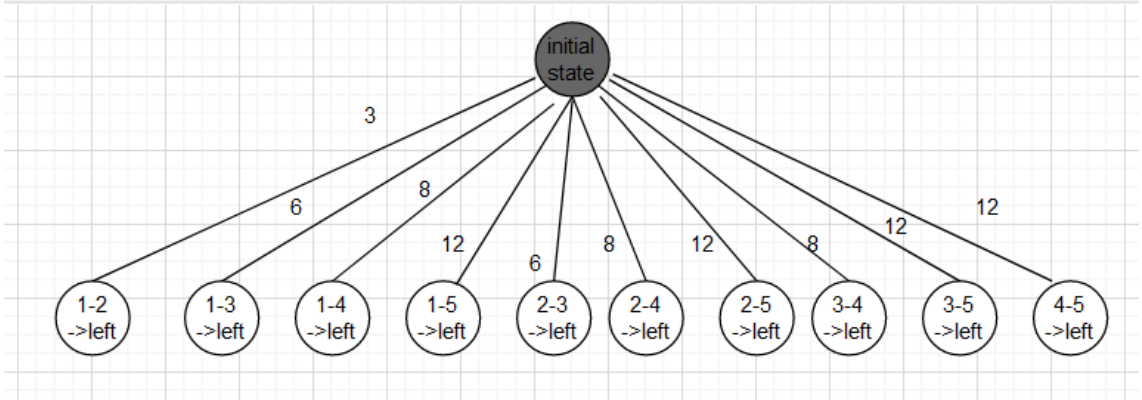
Εισαγωγή

Στα πλαίσια της πρώτης προγραμματιστικής εργασίας του μαθήματος, μεταξύ των δύο projects, η ομάδα μας επέλεξε να ασχοληθεί με το πρόβλημα της Διάσχισης της Γέφυρας. Η λύση του συγκεκριμένου προβλήματος υλοποιήθηκε στην προγραμματιστική γλώσσα της Java, καθώς μας ήταν πιο οικεία σε σχέση με άλλες γλώσσες. Ο κώδικας μας βασίστηκε σε μεγάλο βαθμό σε αυτόν του εργαστηρίου. Παρακάτω γίνεται ανάλυση του αλγορίθμου που χρησιμοποιήθηκε καθώς και επεξήγηση του κώδικα στα βασικά σημεία.



1 Βασική Ιδέα

Για το συγκεκριμένο πρόβλημα, ο αλγόριθμος που βρήκαμε στην πράξη ότι βγάζει σωστό αποτέλεσμα είναι ο A* με κλειστό σύνολο. Θα χρησιμοποιήσουμε το παράδειγμα της παραπάνω εικόνας ώστε να δείξουμε πως ακριβώς δουλεύει ο A*. Τα μέλη της οικογένειας είναι 5 με χρόνους 1,3,6,8,12 αντίστοιχα. Επιθυμητό αποτέλεσμα του αλγορίθμου είναι να περάσουν όλοι απέναντι σε λιγότερο από 30 δευτερόλεπτα βάσει των γνωστών κανόνων του παιχνιδιού. Αρχικά κάθε μέλος βρίσκεται στην δεξιά πλευρά. Αυτή η κατάσταση θα αποτελεί το initial state μας (ρίζα του δέντρου). Οι κινήσεις που μπορούν να συμβούν, είναι είτε να κινηθούν κάποια μέλη προς τα δεξιά (moveRight μέθοδος στον κώδικα) είτε να κινηθούν κάποια μέλη προς τα αριστερά (moveLeft). Μετά από κάθε κίνηση βρισκόμαστε σε ένα διαφορετικό state. Προφανώς στο πρώτο βήμα τα μέλη θα κινηθούν προς τα αριστερά. Επομένως σε πρώτη φάση παράγονται τα παιδιά της ρίζας τα οποία αποτελούν δυνατές κινήσεις από το initial state. Αξίζει να σημειωθεί στο σημείο αυτό, ότι διαπιστώσαμε πως η βέλτιστη λύση, στην κίνηση προς τα αριστερά αναγκαστικά θα περιλαμβάνει 2 άτομα κάθε φορά, καθώς και στην κίνηση προς τα δεξιά μόνο ένα άτομο. Με βάση την προηγούμενη σκέψη, αποφασίσαμε για λόγους απόδοσης-χρόνου στον αλγόριθμό μας (που υλοποιεί τη λογική του A*) να μην συμπεριλάμβάνουμε όλα τα δυνατά παιδιά κάθε φορά από ένα state, αλλά μόνο αυτά τα οποία τηρούν τους παραπάνω περιορισμούς. Αν δηλαδή ξεκινήσουμε έχοντας ως initial state την κατάσταση που βλέπουμε στην εικόνα, μετά από το πρώτο βήμα τα παιδιά που θα παραχθούν θα ναι οι συνδυασμοί 1-2,1-3,1-4,1-5,2-3,2-4,2-5,3-4,3-5,4-5, συνολικά 10 παιδιά.



Η επινόηση της ευρετικής μας ακολούθησε τη λογική της αφαίρεσης περιορισμών, ώστε να πετύχουμε να είναι αποδεκτή. Επίσης το κόστος κάθε μετάβασης (χρόνος που απαιτείται) είναι >0, οπότε θα είναι και συνεπής. Επομένως έχοντας μια συνεπή κι αποδεκτή ευρετική συνάρτηση γνωρίζουμε ότι χρησιμοποιώντας τον A* με κλειστό σύνολο θα έχουμε βέλτιστο αποτέλεσμα, δηλαδή στην περίπτωσή μας τον ελάχιστο χρόνο για να περάσουν όλα τα μέλη απέναντι.

Ο περιορισμός που αφαιρέσαμε είναι το πλήθος των ατόμων που περνάνε τη γέφυρα κάθε φορά. Στο παιχνίδι μπορούν να περνάνε το πολύ 2 άτομα, ενώ τώρα με την εξαπλούστευση που κάναμε θα μπορούν να περνάνε όσοι θέλουν. Η ευρετική μας υπολογίζει το σκορ κάθε παιδιού-κατάστασης ως εξής:

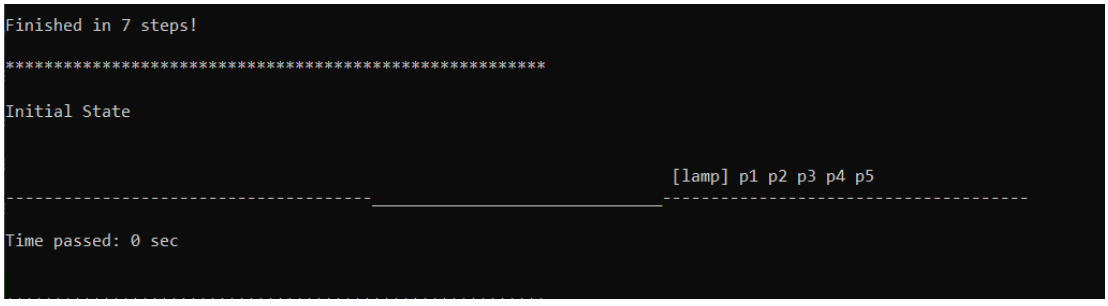
- Αν στην κατάσταση που αξιολογώ, η λάμπα βρίσκεται στην δεξιά πλευρά, το σκορ της συγκεκριμένης κατάστασης ισούται με τον μέγιστο χρόνο από τα άτομα που βρίσκεται δεξιά. Κάτι τέτοιο είναι λογικό, καθώς εφόσον έχουμε αφαιρέσει τον περιορισμό των δύο ατόμων κάθε φορά και περνάνε όσοι θέλουν, για να περάσουν όλοι απέναντι χρειαζόμαστε χρόνο ίσο με του βραδύτερου ατόμου.

- Αν στην κατάσταση που αξιολογώ, η λάμπα βρίσκεται στην αριστερή πλευρά, το σκορ της συγκεκριμένης κατάστασης ισούται με το άθροισμα του ελάχιστου χρόνου από τα άτομα που βρίσκονται αριστερά συν το μέγιστο χρόνο από τα άτομα που βρίσκονται δεξιά. Και πάλι εφόσον δεν υπάρχει πλέον ο περιορισμός των ατόμων, η βέλτιστη λύση για τη συγκεκριμένη κατάσταση είναι να γυρίσει πίσω (δεξιά) ο πιο γρήγορος άνθρωπος και να περάσουν όλοι μαζί στο χρόνο του βραδύτερου.

Η ευρετική μας συνάρτηση συνεπώς υπολογίζει έναν ακέραιο αριθμό που αναπαριστά το σκορ για κάθε κατάσταση, δηλαδή το χρόνο που απαιτείται, βάσει της αφαίρεσης του προαναφερθέντος περιορισμού, ώστε κάθε κατάσταση να φτάσει σε τελική κατάσταση (δηλαδή να είναι όλα τα άτομα αριστερά) με βέλτιστο τρόπο. **Έπειτα μέσω της συνάρτησης evaluate που αναλύεται και παρακάτω, προσθέτουμε αυτό τον ακέραιο αριθμό στο χρόνο που έχει περάσει για κάθε κατάσταση ώστε να φτάσουμε στο συγκεκριμένο σημείο. Έτσι βγαίνει το τελικό σκορ για κάθε κατάσταση-παιδί. Οπότε με αυτό τον τρόπο θα επιλέγουμε να παράγουμε μόνο τα παιδιά της κατάστασης με το καλύτερο σκορ (ελάχιστο στην περίπτωσή μας) κάθε φορά. Εφόσον η ευρετική μας όπως είπαμε έιναι συνεπής κι αποδεκτή τότε σίγουρα η λύση μας θα είναι και βέλτιστη.**

Παρακάτω θα εμφανίσουμε πως το πρόγραμμα που υλοποιήσαμε εμφανίζει γραφικά το βέλτιστο μονοπάτι για το συγκεκριμένο παράδειγμα

Αρχική Κατάσταση



1ο και 2ο βήμα

```
STEP 1: Moved from Right to Left

Person 1 with time: 1 sec
Person 2 with time: 3 sec


[lamp] p1 p2                                                    p3 p4 p5
-----------------------------------_____________________________-----------------------------------

Time passed: 3 sec


**************************************************

STEP 2: Moved from Left to Right

Person 1 with time: 1 sec


p2                                                          [lamp] p3 p4 p5 p1
-----------------------------------_____________________________-----------------------------------

Time passed: 4 sec
```

3ο και 4ο βήμα

```
STEP 3: Moved from Right to Left

Person 3 with time: 6 sec
Person 1 with time: 1 sec


[lamp] p2 p3 p1                                                         p4 p5
-----------------------------------_____________________________-----------------------------------

Time passed: 10 sec


**************************************************

STEP 4: Moved from Left to Right

Person 1 with time: 1 sec


p2 p3                                                       [lamp] p4 p5 p1
-----------------------------------_____________________________-----------------------------------

Time passed: 11 sec
```

5ο και 6ο βήμα

```
STEP 5: Moved from Right to Left

Person 4 with time: 8 sec
Person 5 with time: 12 sec


[lamp] p2 p3 p4 p5                                                      p1
-----------------------------------_____________________________-----------------------------------

Time passed: 23 sec


**************************************************

STEP 6: Moved from Left to Right

Person 2 with time: 3 sec


p3 p4 p5                                                        [lamp] p1 p2
-----------------------------------_____________________________-----------------------------------

Time passed: 26 sec
```

7ο βήμα και Τελική Κατάσταση

Παρατηρούμε ότι για το συγκεκριμένο παράδειγμα ο βέλτιστος χρόνος ειναι 29<30 δευτερόλεπτα.

2 Επεξήγηση-Δομή Κώδικα

Για να υλοποιήσουμε τη λύση του προβλήματος χρησιμοποιήσαμε 4 κλάσεις, τη SpaceSearcher, τη State, την Person και την Main. **Η SpaceSearcher υλοποιεί τη λογική του A* με κλειστό σύνολο και είναι ακριβώς ίδια με αυτήν που υπάρχει στο εργαστήριο, οπότε δεν χρειάζεται περαιτέρω επεξήγηση**. Παρακάτω αναλύονται οι 3 υπόλοιπες κλάσεις.

- **Class Person:** Αυτή η κλάση παριστάνει τον κάθε άνθρωπο. Το κάθε αντικείμενο αυτής θα περιέχει τις μεταβλητές id,time οι οποίες θα αντιστοιχούν στο νούμερο του ατόμου και στο χρόνο που κάνει να περάσει τη γέφυρα. Υπάρχουν 2 κατασκευαστές, ένας ο οποίος δέχεται σαν όρισμα τα 2 αυτά μεγέθη και τα αρχικοποιεί, και ένας που είναι κενός. Εμείς χρησιμοποιούμε στον κώδικα μας μόνο τον πρώτο. Επίσης υπάρχουν και οι κλασσικοί getters και setters ως μέθοδοι για τις μεταβλητές id,time μιας και οι μεταβλητές αυτές έχουν οριστεί ως private.

- **Class State:**Αρχικά στην κλάση κάνουμε implement το Comparable<State> ώστε να μπορέσουμε να κάνουμε override τη μέθοδο compareTo . Σε κάθε κατάσταση έχουμε τη δεξιά πλευρά που περιέχει τους ανθρώπους που βρίσκονται στη δεξιά πλευρά για τη συγκεκριμένη κατάσταση που αναπαρίσταται με ενα arraylist από person ως ArrayList<Person> rightSide,και την αριστερή πλευρά αντίστοιχα που αναπαρίσταται με ArrayList<Person> leftSide. Επίσης χρησιμοποιούμε τις εξής μεταβλητές για κάθε αντικείμενο:

  1. Μεταβλητή lightSide τυπου String ώστε σε ένα συγκεκριμένο state να ξέρουμε σε ποια πλευρά βρίσκεται η λάμπα και έτσι να γνωρίζουμε ποιες είναι οι δυνατές επόμενες κινήσεις.
  2. Μεταβλητή score τύπου int η οποία θα αναπαριστά το σκορ της κάθε κατάστασης, που θα δίνεται μέσω της μεθόδου evaluate που θα δούμε μετέπειτα.
  3. Μεταβλητή father που είναι τύπου State και θα δείχνει στη θέση μνήμης του πατέρα της συγκεκριμένης κατάστασης. Αυτό χρειάζεται κυρίως στο να αναδείξουμε πλήρως το μονοπάτι της βέλτιστης λύσης από τη ρίζα ως την τελική κατάσταση.
  4. Μεταβλητή cost τύπου int που αναπαριστά το χρόνο που έχει περάσει μέχρι τη

συγκεκριμένη κατάσταση. Αυτή η μεταβλητή είναι κρίσιμης σημασίας καθώς, σε αντίθεση με τον bestFS, στον A* μας ενδιαφέρει και το συνολικό κόστος μέχρι να φτάσουμε σε μια κατάσταση (πέρα από το κόστος που δίνει η ευρετική).

Συνεχίζοντας με τη δομή της κλάσης αυτής, στην υλοποίησή μας υπάρχουν 3 κατασκευαστές. Ο πρώτος κατασκευαστής χρησιμοποιείται μόνο για την αρχικοποίηση της ρίζας, οπότε δέχεται ως όρισμα μόνο ένα ArrayList τύπου Person το οποίο δίνεται στη μεταβλητή rightSide μιας και όλα τα μέλη της οικογένειας σε πρώτη φάση είναι στη δεξιά πλευρά. Επίσης με κατάλληλη set μέθοδο που θα αναφέρουμε και πιο κάτω, θέτουμε τη μεταβλητή lightSide να είναι right. Ο δεύτερος κατασκευαστής, δέχεται ορίσματα left,right,lamp,cost ώστε να αναθέσει τις τιμές τους στις αντίστοιχες μεταβλητές που προείπαμε. Τις μεταβλητές father και score τις αλλάζουμε μέσω setter μόνο. Έχουμε και έναν τρίτο κατασκευαστή, ο οποίος είναι κενός σε περίπτωση που απλώς θέλουμε να κατασκευάσουμε ένα αντικείμενο State χωρίς να θέλουμε να κάνουμε κάποια μεταβλητή initialize. Βλέποντας τον κώδικα στη συνέχεια βρίσκουμε τους κλασσικούς setters και getters για κάθε μεταβλητή. Στο σημείο αυτό θα εξηγήσουμε αναλυτικά κάθε μέθοδο της κλάσης πέραν των setters και getters.

- **boolean is Terminal():**Η συγκεκριμένη μέθοδος απλώς τσεκάρει για ένα συγκεκριμένο state αν αποτελεί τελική κατάσταση. Αυτό το ελέγχουμε εύκολα βλέποντας αν η δεξιά πλευρά είναι κενή, με την εντολή return rightSide.isEmpty().
- **int heuristic():** Η μέθοδος αυτή επιστρέφει έναν ακέραιο αριθμό που αξιολογεί την κατάσταση που την καλεί. Αρχικά έχουμε μια if που ελέγχει αν στη συγκεκριμένη κατάσταση, η λάμπα βρίσκεται στη δεξιά πλευρά, δηλαδή αν η μεταβλητή lightSide ειναι ίση με right. Αν κάτι τέτοιο ισχύει, σύμφωνα με τη σκέψη πίσω από την ευρετική μας (έχει αναλυθεί σε πλήρη βαθμό στη σελίδα 3 του παρόντος εγγράφου) αρχικοποιούμε μια μεταβλητή max τύπου int και την κάνουμε ίση με 0, με σκοπό να βρούμε το μέγιστο χρόνο από τα άτομα που βρίσκονται δεξιά, μιας και εφόσον έχει γίνει η αφαίρεση περιορισμού του πλήθους των ατόμων που περνάνε τη γέφυρα, το κόστος της συγκεκριμένης κατάστασης από την τελική ισούται με το χρόνο αυτό. Αυτό επιτυγχάνεται πολύ εύκολα με ένα for loop στο rightside που αποτελεί array με person όπως έχει προαναφερθεί. Με μια if τσεκάρουμε αν υπάρχει χρόνος μεγαλύτερος του παρόντος max και αν ναι τον αναθέτουμε στη μεταβλητή max. Μόλις τελειώσει το for loop επιστρέφουμε το max. Σε διαφορετική περίπτωση, δηλαδή αν η μεταβλητή lightSide ισούται με left, θέλουμε το ελάχιστο χρόνο από τα άτομα που βρίσκονται αριστερά (για να επιστρέψει πίσω με τη λάμπα) συν το μέγιστο από τα άτομα που βρίσκονται δεξιά (συμπεριλαμβανομένου και του ατόμου που επέστρεψε). Με παρόμοιο τρόπο βρίσκουμε το min και max χρόνο από κάθε πλευρά και τελικά επιστρέφουμε το άθροισμά τους.
- **void evaluate()**: Η evaluate σύμφωνα με τον ορισμό του A* θέτει το συνολικό σκορ του συγκεκριμένου state ίσο με το άθροισμα του χρόνου (cost) που έχει περάσει μέχρι τότε και του αποτελέσματος που επιστρέφει η ευρετική.
- **ArrayList<State>getChildren():** Η συγκεκριμένη μέθοδος παράγει τα παιδιά μιας κατάστασης. Τονίζουμε και εδώ ότι ως περιορισμό έχουμε το να μην γίνονται κινήσεις προς τα αριστερά που περιέχουν μόνο 1 άτομο και να μην γίνονται κινήσεις προς τα δεξιά που περιέχουν 2 άτομα, καθώς κάτι τέτοιο είναι προφανές ότι δε θα βγάλει σωστό αποτέλεσμα, οπότε δεν υπάρχει και λόγος να το εξετάσουμε ως πιθανή βέλτιστη λύση. Αρχικοποιούμε σε πρώτη φάση τις μεταβλητές:
  1. **ArrayList<State> children = new ArrayList<State>()**: Αυτη η μεταβλητή είναι ένα array με καταστάσεις που αναπαριστούν τα παιδιά ενός συγκεκριμένου

state που εξερευνώ. Επιστρέφονται για να τοποθετηθουν στο μέτωπο αναζήτησης.

2. **ArrayList<Person> temp = new ArrayList<Person>()**:Αυτή η μεταβλητή είναι προσωρινή, καθώς θα αναπαριστά κάθε φορά τους 2 ανθρώπους που θα πηγαίνουν απο αριστερά προς τα δεξιά. Δηλαδή εξετάζουμε όλους τους δυνατούς συνδυασμούς.
3. **State child**:Αυτή η μεταβλητή θα αναπαριστά το παιδί που παράγουμε κάθε φορά για ένα συγκεκριμένο state.
4. **int maxim=0**: Η συγκεκριμένη μεταβλητή θα χρησιμεύσει στην κίνηση από δεξιά προς τα αριστερά, καθώς όποιος συνδυασμός και να υπάρχει και οι δύο θα κινούνται με το χρόνο του βραδύτερου, οπότε θα βρίσκουμε το μεγαλύτερο εκ των δύο και θα τον αναθέτουμε στη συγκεκριμένη μεταβλητή.
5. **int time1=0**: Στη συγκεκριμένη μεταβλητή θα έχουμε το χρόνο ενός απ των δύο ατόμων που κινούνται αν κινούμαστε προς τα αριστερά και το χρόνο του ενός αν κινούμαστε προς τα δεξιά.
6. **int time2=0**: Στη μεταβλητή αυτή θα αναθέτουμε το χρόνο του δεύτερου ατόμου όταν έχουμε κίνηση προς τα αριστερά.

Η λογική που ακολουθούμε εδώ για να παράγουμε τα παιδιά κάθε φορά, είναι να ελέγχουμε αν το lightSide είναι ίσο με right ή left, μιας και ανάλογα με το που είναι η λάμπα καταλαβαίνουμε και ποιες είναι οι δυνατές επόμενες κινήσεις. Αυτό το καταφέρνουμε πολύ εύκολα με τις εντολές if (lightSide == ”right”) και else if (lightSide == ”left”). Αν ισχύει το πρώτο if σημαίνει ότι η λάμπα βρίσκεται δεξιά. Σε πρώτη φάση τσεκάρουμε μια πολύ ειδική περίπτωση στην οποία έχουμε εξαρχής ένα άτομο, ελέγχοντας ότι αριστερά δεν βρίσκεται κανένας και ότι το μέγεθος της δεξιάς πλευράς είναι 1, με τις εντολές if (this.leftSide.isEmpty() && (this.rightSide.size()==1)). Κατασκευάζουμε ένα ακόμα αντικείμενο state δίνοντας του νέα διεύθυνση μνήμης το οποίο σε πρώτη φάση γίνεται initialized με τα στοιχεία του πατέρα με την εντολή child=new State(this.leftSide,this.rightSide,’right’, this.cost). Στη συνέχεια προφανώς το συγκεκριμένο state θα αλλάξει εφόσον αποτελεί παιδί. Έπειτα για το συγκεκριμένο παιδί βάζουμε τη διεύθυνση του πατέρα του στη μεταβλητή father με την εντολή child.setFather(this). Αυτό είναι ένα σημαντικό σημείο καθώς αν αποτελεί τελική κατάσταση χρειαζόμαστε το μονοπάτι που ακολουθήθηκε ξεκινώντας από τη ρίζα. Στη συνέχεια προσθέτουμε στο κόστος του παιδιού το χρόνο που κάνει το συγκεκριμένο άτομο (μιλάμε μόνο για έναν ακόμα) να περάσει απέναντι με την εντολή child.setCost(this.rightSide.get(0).time). Μιας και έχουμε εξασφαλίσει ότι έχουμε μόνο ένα άτομο στη δεξιά πλευρά, με την παραπάνω εντολή παίρνουμε το χρόνο του και τον προσθέτουμε στο συνολικό κόστος. Προσθέτουμε στο temp array το άτομο αυτό με την εντολή temp.add(this.rightSide.get(0)) και καλούμε την μέθοδο moveLeft με όρισμα το temp που όπως έχουμε πει θα έχει κάθε φορά τους ανθρώπους που εχουν επιλεγεί για να μετακινηθούν(στην περίπτωση αυτή είναι μόνο ένας) για να εκτελέσουμε κίνηση προς τα αριστερά. Η μέθοδος αυτή αναλύεται παρακάτω. Αξίζει να τονίσουμε μιας και επειδή είναι εξαρχής 1 και θα παραχθεί αναγκαστικά μόνο ένα παιδί, δε χρειάζεται να κάνουμε evaluate. Τέλος προσθέτουμε στο array children το συγκεκριμένο παιδί με την εντολή children.add(child). Αν δεν έχουμε εξαρχης ένα άτομο, τότε για το current state η λογική που παράγουμε παιδιά (με τη λάμπα να βρίσκεται στη δεξιά πλευρά) είναι να έχουμε 2 for με τη μια να είναι nested ώστε να βρούμε όλους τους δυνατούς συνδυασμούς με 2 άτομα από τη δεξιά πλευρά που θα κινηθούν προς τα αριστερά, δηλαδή ένα παιδί για κάθε δυνατό συνδυασμό των 2 ατόμων. Αυτό επιτυγχάνεται με τις εντολές for (int i = 0; i < rightSide.size(); i++){

for (int j = i + 1; j < rightSide.size() ; j++). Έπειτα ακολουθούμε ακριβώς την ίδια

λογική με αυτή που είδαμε για το 1 άτομο, με μόνες αλλαγές το ότι βρίσκουμε τους 2 χρόνους με τις εντολές time1=this.rightSide.get(i).getTime() και time2=this.rightSide.get(j).getTime() και μετά βρίσκουμε το μέγιστο χρόνο από τους δύο με την εντολή maxim=Math.max(time1, time2) μιας και θα προσθέσουμε το μέγιστο χρόνο στο κόστος του συγκεκριμένου παιδιού με την εντολή child.setCost(maxim). Επίσης προσθέτουμε στο array temp τα 2 άτομα που έχουν επιλεγεί με την εντολή temp.add(this.rightSide.get(i)) και temp.add(this.rightSide.get(j)). Καλούμε τη moveleft με την εντολή child.moveLeft(temp), κάνουμε evaluate στο παιδί με την εντολή child.evaluate και τέλος προσθέτουμε το παιδί στο array children ώστε να προστεθεί στο μέτωπο αναζήτησης. Με πολύ παρόμοιο τρόπο λειτουργεί το πρόγραμμα και όταν βρίσκεται η λάμπα αριστερά μόνο που στην περίπτωση αυτή δε θα χω εμφωλευμένο for μιας και ο περιορισμός μας είναι ότι θα κινείται προς τα δεξιά μόνο ένα άτομο κάθε φορά.

  - **void moveLeft(ArrayList<Person> people,int max):** Η συγκεκριμένη μέθοδος όπως μπορεί κάποιος να καταλάβει από το όνομά της εκτελεί μία κίνηση προς τα αριστερά. Δέχεται ως όρισμα ένα arraylist με αντικείμενα person, καθώς τα άτομα που θα πάνε δεξιά είναι σίγουρα δύο κάθε φορά, εκτός από την περίπτωση που εξαρχής έχουμε ένα άτομο μόνο που θέλει να περάσει τη γέφυρα. Κάνουμε remove τα άτομα αυτά που έχουν επιλεγεί να μετακινηθούν από το rightSide για το συγκεκριμένο state και τα προσθέτουμε στην αριστερή πλεύρα με την εντολή add μιας και είναι ArrayList. Τέλος για το συγκεκριμένο state κάνουμε τη μεταβλητή lightSide ίσο με left μιας και μετά την κίνηση προς τα αριστερά η λάμπα βρίσκεται πλέον στην αριστερή πλευρά.
  - **void moveRight(Person person):** Στη μέθοδο αυτή μιας και ακολοθούμε τον αρχικό περιορισμό ότι προς τα δεξιά έχει νόημα να κινείται μόνο ένα άτομο κάθε φορά, έχουμε μόνο ένα αντικείμενο Person ως όρισμα και όχι ArrayList. Αντίστοιχα εδώ αφαιρούμε το άτομο αυτό από το leftSide και το προσθέτουμε στο RightSide.
  - **void print(String side):** Μια απλή μέθοδος που μας εκτυπώνει τα άτομα που βρίσκονται σε κάθε πλευρά και τη γέφυρα με έναν κατανοητό τρόπο.

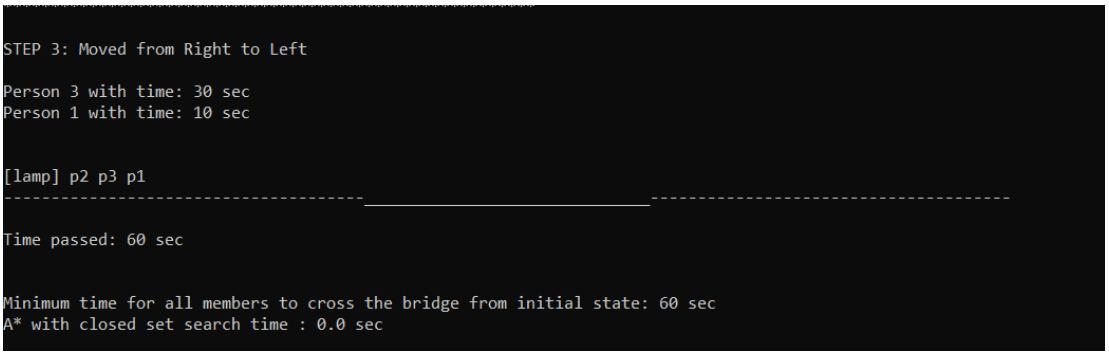
- **Class Main:** Στην κλάση αυτή βρίσκεται η main μας, δηλαδή εκεί που θα τρέξουμε το πρόγραμμα. Αρχικά φτιάχνουμε ένα αντικείμενο Scanner ώστε να μπορούμε να δώσουμε ως είσοδο τον αριθμό των χρηστών το τι χρόνο έχει ο καθένας και το time limiti το οποίο υπάρχει για να περάσουν όλοι οι χρήστες απέναντι. Έπειτα εκτυπώνουμε το μήνυμα Give the number of family members, ώστε να καταλάβει ο χρήστης ότι πρέπει να πληκτρολογήσει τον αριθμό των μελών. Αυτό το επιτυχγάχνουμε με την εντολή int N=in.nextInt(). Μετά φτιάχνουμε ενα arraylist τύπου Person το οποίο ονομάζεται family και θα περιέχει όλα τα μέλη της οικογένειας. Στη συνέχεια γινεται ένα for loop τόσες φορές όσα είναι τα μέλη μας και ζητείται κάθε φορά να δοθεί ο χρόνος για το συγκεκριμένο μέλος. Στο τέλος κάθε for loop φτίαχνουμε ένα αντικείμενο Person δίνοντας το id και το χρόνο του ως όρισμα στον κατασκευαστή και το προσθέτουμε στο arraylist family με την εντολή family.add(temp). Μόλις τελειώσει το for loop κατασκευάζουμε ενα αντικείμενο State με όνομα init_state και δίνουμε ως όρισμα στον κατασκευαστή το family ώστε να γίνει η κατάλληλη αρχικοποίηση στην κατάσταση, δηλαδή να βρίσκονται όλα τα αντικείμενα τύπου Person στο rightSide. Φτιάχνουμε ένα αντικείμενο SpaceSearcher με την εντολή SpaceSearcher space_searcher = new SpaceSearcher() για να μπορέσουμε να εκτελέσουμε τον A* με κλειστό σύνολο. Έπειτα αρχικοποιούμε σε null τη μεταβλητή terminal τύπου State η οποία θα παριστάνει την τελική μας κατάσταση. Μετά κατασκευάζουμε μια μεταβλητή τύπου long που ονομάζεται start με την εντολή long start = System.currentTimeMillis(). Θα τη χρησιμοποιήσουμε για να πάρουμε

την παρούσα ώρα, με σκοπό να βρούμε το συνολικό χρόνο που θα κάνει ο αλγόριθμος. Μετέπειτα καλούμε τη μέθοδο της SpaceSearcher η οποία υλοποιεί επακριβώς τον A* με κλειστό σύνολο με την εντολή terminal = space_searcher.A_StarClosedSet(init_state). Η συγκεκριμένη μέθοδος θα μας επιστρέψει την τελική κατάσταση αν βρήκε κάποια ή null σε διαφορετική περίπτωση. Έπειτα αναθέτουμε την παρούσα ώρα (αφού έχει εκτελεστεί ο αλγόριθμος δηλαδή) στη μεταβλητή end με την εντολή long end = System.currentTimeMillis(). Μετά κάνουμε τον έλεγχο όπου αν η terminal είναι Null σημαίνει ότι δε βρέθηκε λύση και τυπώνεται αντίστοιχο μήνυμα. Αν δεν είναι null σημαίνει ότι βρέθηκε λύση. Σε πρώτη φάση απλώς βρίσκουμε το ακριβές path μέχρι να φτάσουμε σε τελική κατάσταση προσθέτοντας σε ένα arraylist τον πατέρα της κάθε κατάστασης αρχίζοντας από την τελική πρώτα. Μετά απλώς εκτυπώνουμε με ένα σχετικά κατανοητό τρόπο κάθε κατάσταση, δηλαδή το initial state και κάθε επόμενη κίνηση. Το τι εκτυπώνεται λεπτομερώς φαίνεται και στις σελίδες 3-5 όπου παρουσιάζουμε το μονοπάτι που ακολουθήθηκε για ένα συγκεκριμένο παράδειγμα.

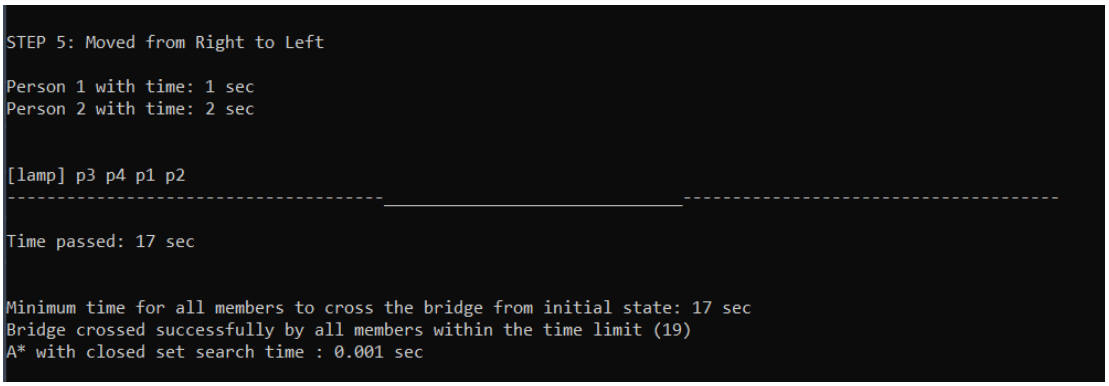
3 Πειραματικά Δεδομένα

Εδώ βλέπουμε τα αποτελέσματα του προγράμματος για διαφορετικό πλήθος και χρόνους ατόμων.

Για N=3 και χρόνους 10,20,30



Για N=4 και χρόνους 1,2,5,10



Για N=5 και χρόνους 4,6,7,9,11

```
STEP 7: Moved from Right to Left

Person 1 with time: 4 sec
Person 2 with time: 6 sec


[lamp] p3 p4 p5 p1 p2
-------------------------------------_____________________________-------------------------------------

Time passed: 44 sec


Minimum time for all members to cross the bridge from initial state: 44 sec
Bridge crossed successfully by all members within the time limit (45)
A* with closed set search time : 0.015 sec
```

Για N=6 και χρόνους 15,16,18,19,24,25

```
STEP 9: Moved from Right to Left

Person 1 with time: 15 sec
Person 2 with time: 16 sec


[lamp] p3 p4 p5 p6 p1 p2
-------------------------------------_____________________________-------------------------------------

Time passed: 154 sec


Minimum time for all members to cross the bridge from initial state: 154 sec
A* with closed set search time : 41.604 sec
```

Για N=7 και χρόνους 1,2,3,4,5,6,7

```
STEP 11: Moved from Right to Left

Person 1 with time: 1 sec
Person 2 with time: 2 sec


[lamp] p4 p5 p3 p6 p7 p1 p2
-------------------------------------_____________________________-------------------------------------

Time passed: 28 sec


Minimum time for all members to cross the bridge from initial state: 28 sec
A* with closed set search time : 35.395 sec
```