Οικονομικό Πανεπιστήμιο Αθηνών
Σχολή Επιστημών και Τεχνολογίας της Πληροφορίας
Τμήμα Πληροφορικής

Τεχνητή Νοημοσύνη

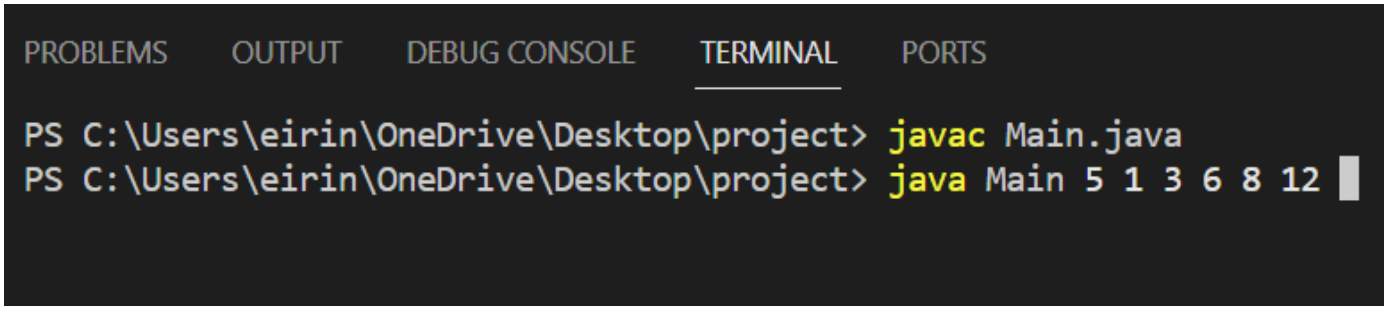
Ειρήνη Κουμαρίνου , p3210085

Μυρτώ Αντωνιάδη , p3210011

Στέφανος Φραγκούλης , p3210212

Το πρόβλημα αφορά έναν αριθμό ατόμων που πρέπει να διασχίσουν μια γέφυρα τη νύχτα με μόνο μία λάμπα και η γέφυρα μπορεί να υποστηρίξει μόνο δύο άτομα κάθε φορά. Κάθε άτομο διασχίζει τη γέφυρα με διαφορετική ταχύτητα και όταν δύο άτομα διασχίζουν μαζί, κινούνται με το ρυθμό του πιο αργού ατόμου. Στόχος μας είναι να βρούμε τον ταχύτερο τρόπο για να περάσουν όλοι οι άνθρωποι τη γέφυρα.

Για την εκτέλεση του προγράμματος ο χρήστης πρέπει να δώσει τον αριθμό των ατόμων και τους αντίστοιχους χρόνους ως ορίσματα της γραμμής εντολών. Το πρώτο όρισμα πρέπει να είναι ο αριθμός των ατόμων. Τα επόμενα ορίσματα αντιπροσωπεύουν το χρόνο που χρειάζεται κάθε άτομο για να διασχίσει τη γέφυρα. Για παράδειγμα , για τις τιμές της εκφώνησης ο χρήστης θα πρέπει να κάνει το εξής :



Σε περίπτωση που ο χρήστης δώσει τιμές που δεν αντιστοιχούν στις προϋποθέσεις του προγράμματος , εμφανίζεται αντίστοιχο μήνυμα λάθους .

Το πρόγραμμα μας δέχεται τιμές από τον χρήστη , σύμφωνα με τις παραπάνω οδηγίες , για τον αριθμό των ατόμων και τους αντίστοιχους χρόνους. Εφαρμόζει μια παραλλαγή του αλγορίθμου αναζήτησης A*. Χρησιμοποιεί ευρετικές λειτουργίες για να δώσει προτεραιότητα στις καταστάσεις που είναι πιθανό να οδηγήσουν σε βέλτιστη λύση. Συγκεκριμένα η ευρετική που χρησιμοποιούμε ,MaxH( ) ,κρατάει το μεγαλύτερο χρόνο της εκάστοτε RightList. Με τη βοήθεια της evaluate( ) υπολογίζουμε το συνολικό κόστος f του κάθε κόμβου και με την σύγκρισή τους επιλέγεται ο επόμενος κόμβος για το μονοπάτι που θα ακολουθήσουμε .

Βασική μας ιδέα είναι η δημιουργία ενός δέντρου που πραγματοποιείται στην μέθοδο getChildren( ) .Πιο συγκεκριμένα για το παράδειγμα που δόθηκε ,στην αρχική μας κατάσταση έχουμε την RightList να περιέχει όλους τους χρόνους , την LeftList κενή και την μεταβλητή lampOnRight αρχικοποιημένη με τιμή True. Με το διπλό for loop παίρνουμε όλες τις πιθανές μετακινήσεις ,ανανεώνουμε τις λίστες καθώς και τη τοποθεσία της λάμπας .Αυτή η διαδικασία συνεχίζεται μέχρι τον τελικό κόμβο . Στο παραπάνω παράδειγμα της αρχικής κατάστασης τα παιδία είναι τα εξής :

RightList[6, 8, 12] RightList[3, 8, 12] RightList[3, 6, 12] RightList[3, 6, 8]

LeftList[1, 3] LeftList[1, 6] LeftList[1, 8] LeftList[1, 12]

RightList[1, 8, 12] RightList[1, 6, 12] RightList[1, 6, 8] RightList[1, 3, 12]

LeftList[3, 6] LeftList[3, 8] LeftList[3, 12] LeftList[6, 8]

RightList[1, 3, 8] RightList[1, 3, 6]

LeftList[6, 12] LeftList[8, 12]

Η μέθοδος print( ) χρησιμοποιείται ώστε να εκτυπώσουμε τις καταστάσεις των δύο λιστών ,τη θέση του φακού και τον χρόνο g. Αρχικά δημιουργούμε ένα αντικείμενο StringBuilder με όνομα output το οποίο κατασκευάζει το τελικό string που θα εκτυπωθεί .Στη συνέχεια, αν η LeftList περιέχει στοιχεία δημιουργούμε ένα string με την επικεφαλίδα "Left side: " και προσθέτουμε τα στοιχεία της λίστας, χωρισμένα με κόμμα και κενό διάστημα. Παρόμοια διαδικασία γίνεται και για τη RightList με επικεφαλίδα "Right side: ". Προσθέτουμε στο string τη θέση του φακού, δηλαδή εάν βρίσκεται στη δεξιά ή την αριστερή πλευρά, με βάση την τιμή της boolean μεταβλητής lampOnRight. Τέλος, με τη μέθοδο this.getG( ) προσθέτουμε τον συνολικό χρόνο και εμφανίζουμε το τελικό αποτέλεσμα με τη χρήση του System.out.println(output).

Στη κλάση Main έπειτα από τους ελέγχους των τιμών που αναφέραμε παραπάνω κατασκευάζουμε την αρχική κατάσταση FirstState με βάση τις αρχικές τιμές (initialTimes) καθώς και τη τερματική κατάσταση EndState , η οποία αρχικά είναι null. Δημιουργούμε δύο λίστες: μία ανοιχτή λίστα (OpenList) για τις εξεταζόμενες καταστάσεις και μία κλειστή λίστα (ClosedList) για τις ήδη εξετασμένες. Η αρχική κατάσταση προστίθεται στην ανοιχτή λίστα και ξεκινάει η while. Επιλέγουμε τη κατάσταση με το χαμηλότερο κόστος από την ανοιχτή λίστα. Αν η τρέχουσα κατάσταση υπερβαίνει το συνολικό χρόνο (sum) η εκτέλεση τερματίζει. Αν η τρέχουσα κατάσταση είναι η τερματική, η επανάληψη σταματά. Σε οποιαδήποτε άλλη περίπτωση προσθέτουμε τα παιδιά της τρέχουσας κατάστασης στην ανοιχτή λίστα, αν δεν έχουν ήδη εξεταστεί και η κατάσταση Curr προστίθεται στη κλειστή λίστα. Αν βρεθεί τερματική κατάσταση, ανασυντάσσεται το μονοπάτι από την τερματική προς την αρχική κατάσταση και εκτυπώνεται. Αν δεν βρεθεί λύση, εμφανίζεται σχετικό μήνυμα.

Το τελικό αποτέλεσμα θα εμφανίζεται ως εξής :

```
PS C:\Users\eirin\OneDrive\Desktop\project> javac Main.java
PS C:\Users\eirin\OneDrive\Desktop\project> java Main 5 1 3 6 8 12
Right side: 1, 3, 6, 8, 12 Torch position: Right Time taken: 0
Left side: 1, 3 Right side: 6, 8, 12 Torch position: Left Time taken: 3
Left side: 3 Right side: 6, 8, 12, 1 Torch position: Right Time taken: 4
Left side: 3, 6, 1 Right side: 8, 12 Torch position: Left Time taken: 10
Left side: 3, 6 Right side: 8, 12, 1 Torch position: Right Time taken: 11
Left side: 3, 6, 8, 12 Right side: 1 Torch position: Left Time taken: 23
Left side: 6, 8, 12 Right side: 1, 3 Torch position: Right Time taken: 26
Left side: 6, 8, 12, 1, 3 Torch position: Left Time taken: 29
PS C:\Users\eirin\OneDrive\Desktop\project>
```

Ένα άλλο πειραματικό παράδειγμα με 3 άτομα με χρόνους 1, 3, 5 αντίστοιχα έχει το εξής αποτέλεσμα :

```
PS C:\Users\eirin\OneDrive\Desktop\project> javac Main.java
PS C:\Users\eirin\OneDrive\Desktop\project> java Main 3 1 3 5
Right side: 1, 3, 5 Torch position: Right Time taken: 0
Left side: 1, 3 Right side: 5 Torch position: Left Time taken: 3
Left side: 3 Right side: 5, 1 Torch position: Right Time taken: 4
Left side: 3, 5, 1 Torch position: Left Time taken: 9
PS C:\Users\eirin\OneDrive\Desktop\project>
```