Μητσάκης Νίκος - p3210122
Παντελίδης Ιπποκράτης - p3210150
Καμπούρης Βαγγέλης – p3210061
ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ 2023-2024

Αναφορά Παράδοσης

1η Εργασία στην Τεχνητή Νοημοσύνη – Διάσχιση Ποταμού

Η εργασία μας αποτελείται από τα παρακάτω αρχεία Java:

FamilyMember.java

Όνομα Κλάσης: FamilyMember

Γνωρίσματα Κλάσης:

- private String name: αποθηκεύει το όνομα του μέλους της οικογένειας.
- private int time: αποθηκεύει τον χρόνο που απαιτείται από το μέλος της οικογένειας για να διασχίσει τον ποταμό.

Μέθοδοι:

- Κατασκευαστής (Constructor): αρχικοποιεί το όνομα και τον χρόνο διάσχισης του μέλους της οικογένειας.
- Setters and Getters
- Overriden toString: χρησιμοποιείται για την εμφάνιση των στοιχείων ενός ατόμου της οικογένειας σε εκτυπώσιμη μορφή.

Περιγραφή: Αναπαριστά ένα άτομο της οικογένειας που προσπαθεί να διασχίσει το ποτάμι. Λειτουργεί ως μια βασική δομή δεδομένων για την αποθήκευση πληροφοριών σχετικά με κάθε μέλος της οικογένειας που συμμετέχει στο παιχνίδι διάσχισης του ποταμού, όπως το όνομα και ο απαιτούμενος χρόνος για τη διάσχιση.

State.java

Όνομα Κλάσης: State

Γνωρίσματα Κλάσης:

- private List<FamilyMember> leftBank : Αντιπροσωπεύει την αριστερή όχθη του ποταμού και ποια άτομα βρίσκονται εκεί σε κάθε κατάσταση.
- private List<FamilyMember> rightBank : Αντιπροσωπεύει την δεξιά όχθη του ποταμού και ποια άτομα βρίσκονται εκεί σε κάθε κατάσταση.
- private boolean lanternOnRightBank : Αντιπροσωπεύει την τοποθεσία του φαναριού στον κόσμο. Αν είναι true, το φανάρι βρίσκεται στην δεξιά όχθη, ενώ αν είναι false βρίσκεται στην αριστερή.

- private int cost : Το πραγματικό (χρονικό) κόστος που έχει καταγραφεί για να φτάσουμε μέχρι αυτή την κατάσταση.
- private int heuristic : Η εκτίμηση του κόστους μέχρι την τελική κατάσταση που μας δίνει η ευρετική συνάρτηση.
- private int totalCost : Το υποτιθέμενο μέγιστο κόστος της συγκεκριμένης κατάστασης.
- private State father : Ουσιαστικά είναι μία αναφορά που μας δείχνει από ποια κατάσταση 'γεννήθηκε' η συγκεκριμένη κατάσταση (χρήσιμο στην αναζήτηση/διάσχιση του δέντρου).

Μέθοδοι:

- Hashcode() : Παράγει ένα μοναδικό αναγνωριστικό για να χρησιμοποιηθεί ως id κάθε κατάστασης.
- Equals(Object) : Ελέγχει αν δύο καταστάσεις είναι «ίδιες/ισοδύναμες», δηλαδή αν περιγράφουν το ίδιο στιγμιότυπο του προγράμματος. Συγκεκριμένα ελέγχει αν οι δύο λίστες που περιγράφουν ποια άτομα της οικογένειας βρίσκονται σε κάθε όχθη περιέχουν τα ίδια άτομα και επιπλέον αν η μεταβλητή που δηλώνει σε ποια όχθη βρίσκεται το φανάρι έχει την ίδια τιμή και στις δύο καταστάσεις.
- compareTo(State) : Συγκρίνει δύο καταστάσεις, βάσει το υποτιθέμενο συνολικό κόστος τους και επιστρέφει 0 αν είναι ίσες, 1 αν η συγκεκριμένη κατάσταση είναι 'μεγαλύτερη' της κατάστασης στο γνώρισμα της συνάρτησης, ενώ επιστρέφει -1 αν ισχύει το αντίθετο.
- getChildren() : Δημιουργεί καταστάσεις παιδιά, προσομοιώνοντας όλες τις δυνατές κινήσεις από την τρέχουσα κατάσταση, λαμβάνοντας υπόψη και τις ατομικές, αλλά και τις διασχίσεις ανά ζευγάρια. Επιστρέφει μια λίστα αυτών των καταστάσεων, κάθε μία συσχετιζόμενη με την κατάσταση γονέα της.
- crossRiver(FamilyMember, FamilyMember) : Πραγματοποιεί μία διάσχιση του ποταμού. Υπολογίζει μια νέα κατάσταση μετά τη διάσχιση ενός ή δύο μελών της οικογένειας από μία όχθη στην άλλη.
- calculateHeuristic2() : Εκτιμάει πόσο θα κοστίσει (χρονικά) από την συγκεκριμένη κατάσταση να καταλήξουμε σε κάποια τελική κατάσταση.
- isFinalState() : Ελέγχει και επιστρέφει true ή false ανάλογα με το αν η συγκεκριμένη κατάσταση είναι τελική, το οποίο σημαίνει ότι δεν βρίσκεται κανένα άτομο στη δεξιά όχθη του ποταμού, δηλαδή η λίστα rightBank είναι άδεια.
- isLanternOnRightBank() : Ελέγχει και επιστρέφει true ή false ανάλογα με το αν το φανάρι στην συγκεκριμένη κατάσταση βρίσκεται στην δεξιά όχθη του ποταμού, δηλαδή αν η μεταβλητή lanternOnRightBank έχει τιμή true.
- Setters and Getters

Περιγραφή:

- **Υλοποίηση του Interface Comparable:**
  Η κλάση State υλοποιεί το interface comparable, το οποίο δημιουργεί μία ταξινόμηση/ιεραρχία μεταξύ αόριστων αντικειμένων, όπως είναι και τα αντικείμενα State . Για τον λόγο αυτό, κάνουμε overload τις μεθόδους compareTo, equals και hashCode.

Μητσάκης Νίκος - p3210122
Παντελίδης Ιπποκράτης - p3210150
Καμπούρης Βαγγέλης – p3210061

ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ 2023-2024

- **Κατασκευαστής (Constructor):**
  Η κατασκευή ενός αντικειμένου State μπορεί να γίνει θεωρητικά με 3 τρόπους, αν και ο default constructor χωρίς ορίσματα που κατασκευάζει μία «άδεια» κατάσταση δεν χρησιμοποιείται ποτέ. Ο overloaded contructor που χρησιμοποιούμε στο πρόγραμμά μας, αρχικά δέχεται τα παρακάτω εξής ορίσματα: leftBank και rightBank, lanternOnRightBank και cost. Με βάση αυτά τα γνωρίσματα αρχικοποιούνται κατάλληλα τα αντίστοιχα πεδία του αντικειμένου State και επιπλέον υπολογίζεται το ευρετικό κόστος της κατάστασης, καλώντας την κατάλληλη συνάρτηση και τέλος τα δύο αυτά κόστη προστίθενται και το αποτέλεσμά τους αποθηκεύεται στο πεδίο totalCost του νεοδημιουργηθέντος αντικειμένου. Επιπλέον, χρησιμοποιούμε και τον copy constructor (ο οποίος δέχεται ως όρισμα ένα υπάρχον State) για να διατηρήσουμε αμετάβλητη την τρέχουσα κατάσταση όταν καλούμε την συνάρτηση get children για να βρούμε τις πιθανές καταστάσεις παιδιά της τρέχουσας.
- **Συνάρτηση Υπολογισμού Νέων Καταστάσεων:**
  Η συνάρτηση getChildren δέχεται ως όρισμα μία υπάρχουσα κατάσταση (αντικείμενο State) και επιστρέφει μία λίστα που περιέχει όλες τις πιθανές καταστάσεις που δύναται να προκύψουν από αυτή ανάλογα με το ποιο ή ποια άτομα της οικογένειας θα διασχίσουν το ποτάμι. Αξίζει να αναφερθεί, ότι πριν από κάθε εξέταση κάποιας πιθανής κίνησης (για παράδειγμα, η διάσχιση του ποταμού ενός ή δύο μελών της οικογένειας), είναι σημαντικό να μην επηρεαστεί η αρχική κατάσταση του παιχνιδιού. Αυτό εξασφαλίζεται δημιουργώντας ένα αντίγραφο της τρέχουσας κατάστασης πριν από κάθε δοκιμαστική κίνηση. Επιπροσθέτως, καθώς το πρόγραμμα δοκιμάζει διάφορες κινήσεις και συνδυασμούς, η αρχική κατάσταση του παιχνιδιού πρέπει να παραμένει αμετάβλητη, ώστε να μπορεί να χρησιμοποιηθεί ως βάση για τις επόμενες δοκιμές.
- **Βασική Συνάρτηση Διάσχισης:**
  Η συνάρτηση crossRiver επιστρέφει την νέα κατάσταση του παιχνιδιού καλώντας κατάλληλα κάποια από τις βοηθητικές συναρτήσεις διάσχισης ανάλογα με το αν έχουμε διάσχιση ενός ή δύο ατόμων και αν η διάσχιση γίνεται από αριστερά προς τα δεξιά ή το ανάποδο, εφόσον έχει πρώτα επιβεβαιώσει ότι πληρούνται οι προυποθέσεις για την διάσχιση (έγκυρα αντικείμενα FamilyMember).
- **Βοηθητικές Συναρτήσεις Διάσχισης:**
  singleToLeftCross -> Υπολογίζει μια νέα κατάσταση για διάσχιση ενός ατόμου από δεξιά προς αριστερά. Για να πραγματοποιηθεί η διάσχιση απαιτεί το άτομο που θέλει να διασχίσει το ποτάμι να βρίσκεται στην δεξιά όχθη, όπως και το φανάρι.
  pairToLeftCross -> Υπολογίζει μια νέα κατάσταση για μία διάσχιση δύο ατόμων από δεξιά προς αριστερά. Για να πραγματοποιηθεί η διάσχιση απαιτεί τα άτομα που θέλουν να διασχίσουν το ποτάμι να βρίσκονται στην δεξιά όχθη, όπως και το φανάρι.
  singleToRightCross -> Υπολογίζει μια νέα κατάσταση για μία διάσχιση ενός ατόμου από αριστερά προς δεξιά. Για να πραγματοποιηθεί η διάσχιση απαιτεί το άτομο που θέλει να διασχίσει το ποτάμι να βρίσκεται στην αριστερή όχθη, όπως και το φανάρι.

pairToRightCross -> Υπολογίζει μια νέα κατάσταση για διάσχιση δύο ατόμων από αριστερά προς δεξιά. Για να πραγματοποιηθεί η διάσχιση απαιτεί τα άτομα που θέλουν να διασχίσουν το ποτάμι να βρίσκονται στην αριστερή όχθη, όπως και το φανάρι.

- **Ευρετικές Συναρτήσεις:**
  - calculateHeuristic(): Η πρώτη ευρετική που σκεφτήκαμε, η οποία βρίσκει το χρόνο που χρειάζεται το βραδύτερο άτομο που βρίσκεται στην δεξιά όχθη να διασχίσει το ποτάμι. Η συγκεκριμένη ευρετική είναι αποδεκτή, διότι υποεκτιμά πάντα το κόστος που θα χρειαστεί για να φτάσουμε στην τελική κατάσταση. Αγνοεί την ύπαρξη του φαναριού, όπως και των διασχίσεων προς τα πίσω για να δώσουμε το φανάρι στους υπόλοιπους που θέλουν να περάσουν απέναντι. Δεν την χρησιμοποήσαμε τελικά γιατί ήταν υπερβολικά γενική και υποεκτιμούσε κατά πολύ την απόσταση κάθε κατάστασης από την τελική.
  - **calculateHeuristic2():** Επεκτείνοντας το σκεπτικό της πρώτης ευρετικής, γράψαμε μία δεύτερη, την οποία και χρησιμοποιήσαμε τελικά στο πρόγραμμά μας. Η συγκεκριμένη ευρετική βρίσκει το υποτιθέμενο κόστος που θα χρειαστούμε για να φτάσουμε από την τρέχουσα κατάσταση σε κάποια τελική. Πρώτιστα ελέγχει αν η τρέχουσα κατάσταση είναι η αρχική οπότε και αφαιρεί από το ήδη γνωστό άθροισμα όλων των χρόνων διάσχισης των μελών της οικογένειας το χρόνο του γρηγορότερου μέλους επί το πλήθος των ζευγαριών που μπορούν να δημιουργηθούν από τη συγκεκριμένη οικογένεια. Έπειτα, αν η τρέχουσα κατάσταση δεν είναι η αρχική, ελέγχει εάν έχουμε φτάσει στην τελική κατάσταση, οπότε και επιστρέφει κατευθείαν τιμή 0, καθώς δεν έχουμε να διανύσουμε άλλη απόσταση, αφού έχουμε ήδη φτάσει στον στόχο μας. Αν δεν είναι τίποτα από τα δύο ο υπολογισμός της ευρετικής τιμής αρχίζει υπολογίζοντας το χρόνο που χρειάζεται το βραδύτερο άτομο που βρίσκεται στην δεξιά όχθη να διασχίσει το ποτάμι. Συνεχίζει με το να πολλαπλασιάσει τον αριθμό αυτόν με το πλήθος των ζευγαριών που μπορούν να δημιουργηθούν από τα άτομα που βρίσκονται στη δεξιά όχθη του ποταμιού. Έπειτα, προσθέτει στην ευρετική τιμή τον αριθμό των απαιτούμενων διαδρομών επιστροφής, προσαρμόζοντάς τον ανάλογα με το αν ο αριθμός των μελών της οικογένειας στη δεξιά όχθη είναι περιττός. Τέλος, για να αντιπροσωπευτούν καλύτερα οι μη-τελικές καταστάσεις στις οποίες το φανάρι βρίσκεται στην αριστερή όχθη, για καταστάσεις στις οποίες η μεταβλητή isLanternOnrightBank συμφωνεί με τη μεταβλητή false υπολογίζεται ο χρόνος που θα χρειαστεί το ταχύτερο άτομο στην όχθη αυτή για να φέρει το φανάρι στην δεξιά όχθη και προστίθεται στην ευρετική τιμή. Η συγκεκριμένη ευρετική είναι αποδεκτή, διότι υποεκτιμά πάντα το κόστος που θα χρειαστεί για να φτάσουμε στην τελική κατάσταση.

Μητσάκης Νίκος - p3210122
Παντελίδης Ιπποκράτης - p3210150
Καμπούρης Βαγγέλης – p3210061

ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ 2023-2024

SpaceSearcher.java

Όνομα Κλάσης: SpaceSearcher

Γνωρίσματα Κλάσης:

- private PriorityQueue<State> frontier: Αντιπροσωπεύει το μέτωπο αναζήτησης, που θα χρησιμοποιηθεί από τον αλγόριθμο A* για να αποθηκεύει τις ανεξευρέυνητες καταστάσεις. Έχει υλοποιηθεί χρησιμοποιώντας μία ουρά προτεραιότητας, βάσει του totalCost πεδίου των αντικειμένων State.
- private HashSet<State> closedSet: Αντιπροσωπεύει το κλειστό σύνολο που θα εμπεριέχει όσες καταστάσεις έχουν εξερευνηθεί από τον αλγόριθμο A*, με σκοπό να αποφευχθεί η παγίδευσή του. Έχει υλοποιηθεί χρησιμοποιώντας ένα hash set.
- public static int timePassed: Μεταβλητή που αποθηκεύει τον συνολικό χρόνο που προσπαθεί να περάσει όλη η οικογένεια στην αριστερή όχθη του ποταμού. Αρχικοποιείται με την τιμή 0.

Μέθοδοι:

- State AStarAlgorithm(State initialState): Υλοποίηση του αλγορίθμου A*.
- Static void retrievePath(State finalState): Ανακτά και εκτυπώνει το μονοπάτι από την αρχική μέχρι την τελική κατάσταση (αν υπάρχει).
- private static void printPath(ArrayList<State> path): Εμφανίζει το μονοπάτι από την αρχική προς την τελική κατάσταση.

Περιγραφή:

- Κατασκευαστής (constructor): Απλά αρχικοποιεί τις μεταβλητές frontier και closedSet.
- **Υλοποίηση του αλγορίθμου A*:** Ξεκινάει με την προσθήκη της αρχικής κατάστασης στην ουρά προτεραιότητας frontier και συνεχίζει με την εξαγωγή και εξέταση της πρώτης κατάστασης της frontier, δηλαδή της κατάστασης με το μικρότερο συνολικό κόστος από όλο το μέτωπο αναζήτησης κάθε φορά, μέχρι να βρεθεί η τελική κατάσταση ή να εξαντληθεί το χρονικό περιθώριο που έχει οριστεί. Κάθε φορά που πάει να εξερευνήσει μία κατάσταση, ελέγχει ότι δεν είναι αντίγραφο κάποιας που έχει εξερευνήσει ήδη κοιτώντας αν υπάρχει στο closedSet. Αν υπάρχει ήδη απλά την αφαιρεί από τη frontier και συνεχίζει στην επόμενη, ενώ αν δεν υπάρχει την προσθέτει στο closedSet και έπειτα την εξερευνεί.

Μητσάκης Νίκος - p3210122
Παντελίδης Ιπποκράτης - p3210150
Καμπούρης Βαγγέλης – p3210061

ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ 2023-2024

UserInputValidator.java

Όνομα Κλάσης: UserInputValidator

Γνωρίσματα Κλάσης:

- public final static String REGEX_INT: Στατική σταθερά που ορίζει το regex (regular expression-κανονική έκφραση) για την επικύρωση ακέραιων αριθμών.

Μέθοδοι:

- public static int validateInputInt(String input, String label): Επικυρώνει αν η είσοδος είναι έγκυρος ακέραιος αριθμός. Πρώτα ελέγχει αν η είσοδος είναι άδεια, στη συνέχεια αν ταιριάζει με το regex για ακέραιους αριθμούς και τέλος αν ο αριθμός αυτός είναι μεγαλύτερος από το μηδέν. Αν η είσοδος δεν πληροί κάποιον από αυτούς τους ελέγχους, το πρόγραμμα τερματίζεται.
- public static String validateInputString(String input, String label): Επικυρώνει αν η είσοδος είναι μη κενό string. Εάν η είσοδος είναι κενή, το πρόγραμμα τερματίζεται. Αλλιώς, επιστρέφει την είσοδο μετά την αφαίρεση τυχόν προπορευόμενων ή ακολουθουμένων κενών χαρακτήρων.

Main.java

Όνομα Κλάσης: Main

Γνωρίσματα Κλάσης:

- public static int totalTime : Στατική μεταβλητή που αποθηκεύει τον συνολικό χρόνο που είναι διαθέσιμος για το παιχνίδι.
- public static int counterOfFamilyMembers : Στατική μεταβλητή που αποθηκεύει τον αριθμό των μελών της οικογένειας.
- public static int sumTime : Στατική μεταβλητή που αποθηκεύει το άθροισμα των χρόνων διάσχισης όλων των μελών της οικογένειας.
- public static List<FamilyMember> rightBank : Στατική λίστα που αποθηκεύει τα μέλη της οικογένειας που βρίσκονται στην δεξιά όχθη του ποταμού.
- public static List<FamilyMember> leftBank : Στατική λίστα που αποθηκεύει τα μέλη της οικογένειας που βρίσκονται στην αριστερή όχθη του ποταμού.

Μέθοδοι:

- public static String readFile(String filePath) : Μέθοδος για την ανάγνωση και επιστροφή του περιεχομένου ενός αρχείου.
- public static void main(String[] args) : Αναλαμβάνει την ανάγνωση των δεδομένων από ένα αρχείο, τη δημιουργία της αρχικής κατάστασης του παιχνιδιού και την εκκίνηση της διαδικασίας επίλυσης μέσω του αλγορίθμου A*. Επιπλέον, εμφανίζει το χρόνο εκτέλεσης

του προγράμματος, τα αποτελέσματα και το μονοπάτι που ακολουθήθηκε (δεδομένου ότι βρέθηκε λύση).

Περιγραφή:

- Η λογική της κύριας μεθόδου (main) είναι η εξής: Αρχικά ελέγχει αν έχει δοθεί όνομα αρχείου ως είσοδος. Στη συνέχεια, διαβάζει τα δεδομένα από το αρχείο και διασπά το περιεχόμενο σε γραμμές. Κάθε γραμμή αντιστοιχεί είτε στον συνολικό διαθέσιμο χρόνο είτε σε ένα μέλος της οικογένειας και τον απαιτούμενο χρόνο για τη διάσχιση. Αφού διαβάσει όλα τα δεδομένα, δημιουργεί την αρχική κατάσταση και ξεκινά την αναζήτηση για τη βέλτιστη λύση χρησιμοποιώντας την κλάση SpaceSearcher. Τέλος, εμφανίζει τη διαδρομή που ακολουθήθηκε (αν βρέθηκε λύση) και τον συνολικό χρόνο αναζήτησης.

Αποτελέσματα δοκιμών διαφόρων αρχείων εισόδου

[Laptop που χρησιμοποιήθηκε για τις δοκιμές: Lenovo Thinkbook 15 G3 ACL { CPU = AMD Ryzen 7 5700U, M = 16 GB }]

- C_n: πραγματικό κόστος/απόσταση της τρέχουσας κατάστασης n μέχρι την τελική. Υπολογίζεται ως sol - g_n , όπου sol το κόστος της βέλτιστης λύσης και g_n το κόστος που έχει καταγραφεί από την αρχική έως την τωρινή κατάσταση
- h_n: εκτίμηση κόστους/απόστασης της τρέχουσας κατάστασης n μέχρι την τελική.

- Αρχείο εισόδου: input_1.txt ***[Αρχείο με τα στοιχεία του βασικού Παραδείγματος]***
- Εύρος Χρόνου Αναζήτησης: μεταξύ 0.003 και 0.009 δευτερόλεπτα
- Όριο Χρόνου: 30
- Πλήθος ατόμων: 4
- Χρόνοι Διάσχισης κάθε μέλους της οικογένειας:
  - Boy: 1
  - Teen: 3
  - Man: 6
  - Woman: 10
  - Granny: 12
- Κόστος (βέλτιστης) λύσης που βρίσκει ο αλγόριθμος: 29
- Η ευρετική συνάρτηση υποεκτιμάει το κόστος σε κάθε βήμα:
  - StartState: $h_0 = 28$, $C_0 = 29 – 0 = 29$ | $h_0 < C_0$
  - State1: $h_1 = 15$, $C_1 = 29 – 3 = 26$ | $h_1 < C_1$
  - State2: $h_2 = 24$, $C_2 = 29 – 4 = 25$ | $h_2 < C_2$
  - State3: $h_3 = 9$, $C_3 = 29 – 16 = 13$ | $h_3 < C_3$
  - State4: $h_4 = 8$, $C_4 = 29 – 19 = 10$ | $h_4 < C_4$
  - State5: $h_5 = 3$, $C_5 = 29 – 22 = 7$ | $h_5 < C_5$
  - State6: $h_6 = 6$, $C_6 = 29 – 23 = 6$ | $h_6 <= C_6$
  - FinalState: $h_7 = 0$, $C_7 = 29 – 29 = 0$ | $h_7 <= C_7$

Μητσάκης Νίκος - p3210122
Παντελίδης Ιπποκράτης - p3210150
Καμπούρης Βαγγέλης – p3210061

ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ 2023-2024

- Αρχείο εισόδου: input_01.txt
- Εύρος Χρόνου Αναζήτησης: μεταξύ 0.003 και 0.009 δευτερόλεπτα
- Όριο Χρόνου: 23
- Πλήθος ατόμων: 4
- Χρόνοι Διάσχισης κάθε μέλους της οικογένειας:
  - Kid: 2
  - Teenager: 4
  - Adult: 5
  - Elder: 10
- Κόστος (βέλτιστης) λύσης που βρίσκει ο αλγόριθμος: 23
- Η ευρετική συνάρτηση υποεκτιμάει το κόστος σε κάθε βήμα:
  - StartState: $h_0 = 17$, $C_0 = 23 – 0 = 23$ | $h_0 < C_0$
  - State1: $h_1 = 7$, $C_1 = 23 – 10 = 13$ | $h_1 < C_1$
  - State2: $h_2 = 7$, $C_2 = 23 – 12 = 11$ | $h_2 < C_2$
  - State3: $h_3 = 4$, $C_3 = 23 – 16 = 7$ | $h_3 < C_3$
  - State4: $h_4 = 5$, $C_4 = 23 – 18 = 5$ | $h_4 <= C_4$
  - FinalState: $h_5 = 0$, $C_5 = 23 – 23 = 0$ | $h_5 <= C_5$

- Αρχείο εισόδου: input_02.txt
- Εύρος Χρόνου Αναζήτησης: μεταξύ 0.004 και 0.012 δευτερόλεπτα
- Όριο Χρόνου: 40
- Πλήθος ατόμων: 5
- Χρόνοι Διάσχισης κάθε μέλους της οικογένειας:
  - Girl: 3
  - Boy: 5
  - Mother: 8
  - Father: 6
  - Grandfather: 12
- Κόστος (βέλτιστης) λύσης που βρίσκει ο αλγόριθμος: 39
- Η ευρετική συνάρτηση υποεκτιμάει το κόστος σε κάθε βήμα:
  - StartState: $h_0 = 28$, $C_0 = 39 – 0 = 0$ | $h_0 < C_0$
  - State1: $h_1 = 17$, $C_1 = 39 – 5 = 34$ | $h_1 < C_1$
  - State2: $h_2 = 24$, $C_2 = 39 – 8 = 31$ | $h_2 < C_2$
  - State3: $h_3 = 11$, $C_3 = 39 – 20 = 19$ | $h_3 < C_3$
  - State4: $h_4 = 8$, $C_4 = 39 – 25 = 14$ | $h_4 < C_4$
  - State5: $h_5 = 5$, $C_5 = 39 – 30 = 9$ | $h_5 < C_5$
  - State6: $h_6 = 6$, $C_6 = 39 – 33 = 6$ | $h_6 <= C_6$
  - FinalState: $h_7 = 0$, $C_7 = 39 – 39 = 0$ | $h_7 <= C_7$

Μητσάκης Νίκος - p3210122
Παντελίδης Ιπποκράτης - p3210150
Καμπούρης Βαγγέλης – p3210061

ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ 2023-2024

- Αρχείο εισόδου: input_03.txt
- Εύρος Χρόνου Αναζήτησης: μεταξύ 0.016 και 0.032 δευτερόλεπτα
- Όριο Χρόνου: 45
- Πλήθος ατόμων: 6
- Χρόνοι Διάσχισης κάθε μέλους της οικογένειας:
  - Toddler: 1
  - Brother: 4
  - Sister: 6
  - Uncle: 9
  - Aunt: 7
  - Grandma: 15
- Κόστος (βέλτιστης) λύσης που βρίσκει ο αλγόριθμος: 43
- Η ευρετική συνάρτηση υποεκτιμάει το κόστος σε κάθε βήμα:
  - StartState: $h_0 = 39$, $C_0 = 43 – 0 = 43$ | $h_0 < C_0$
  - State1: $h_1 = 31$, $C_1 = 43 – 6 = 37$ | $h_1 < C_1$
  - State2: $h_2 = 32$, $C_2 = 43 – 7 = 36$ | $h_2 < C_2$
  - State3: $h_3 = 18$, $C_3 = 43 – 11 = 32$ | $h_3 < C_3$
  - State4: $h_4 = 30$, $C_4 = 43 – 12 = 31$ | $h_4 < C_4$
  - State5: $h_5 = 16$, $C_5 = 43 – 19 = 24$ | $h_5 < C_5$
  - State6: $h_6 = 17$, $C_6 = 43 – 20 = 23$ | $h_6 < C_6$
  - State7: $h_7 = 6$, $C_7 = 43 – 35 = 8$ | $h_7 < C_7$
  - State8: $h_8 = 4$, $C_8 = 43 – 39 = 4$ | $h_8 <= C_8$
  - FinalState: $h_9 = 0$, $C_9 = 43 – 43 = 0$ | $h_9 <= C_9$

- Αρχείο εισόδου: input_04.txt
- Εύρος Χρόνου Αναζήτησης: μεταξύ 0.014 και 0.02 δευτερόλεπτα
- Όριο Χρόνου: 133
- Πλήθος ατόμων: 10
- Χρόνοι Διάσχισης κάθε μέλους της οικογένειας:
  - Toddler 1
  - Boy 2
  - Girl 2
  - Teen 3
  - Man 6
  - Woman 8
  - Grandma 12
  - Grandad 24
  - GreatGrandad 40
  - GreatGrandma 35
- Κόστος (βέλτιστης) λύσης που βρίσκει ο αλγόριθμος: 105
- Η ευρετική συνάρτηση υποεκτιμάει το κόστος σε κάθε βήμα:

Μητσάκης Νίκος - p3210122
Παντελίδης Ιπποκράτης - p3210150
Καμπούρης Βαγγέλης – p3210061

- StartState: $h_0 = 128$, $C_0 = 105 – 0 = 105$ | $h_0 < C_0$
- State1: $h_1 = 161$, $C_1 = 105 – 2 = 103$ | $h_1 < C_1$
- State2: $h_2 = 101$, $C_2 = 105 – 3 = 102$ | $h_2 < C_2$
- State3: $h_3 = 76$, $C_3 = 105 – 43 = 62$ | $h_3 < C_3$
- State4: $h_4 = 96$, $C_4 = 105 – 45 = 60$ | $h_4 < C_4$
- State5: $h_5 = 37$, $C_5 = 105 – 69 = 36$ | $h_5 < C_5$
- State6: $h_6 = 38$, $C_6 = 105 – 70 = 35$ | $h_6 < C_6$
- State7: $h_7 = 19$, $C_7 = 105 – 82 = 23$ | $h_7 < C_7$
- State8: $h_8 = 24$, $C_8 = 105 – 83 = 24$ | $h_8 < C_8$
- State9: $h_9 = 17$, $C_9 = 105 – 85 = 20$ | $h_9 < C_9$
- State10: $h_{10} = 18$, $C_{10} = 105 – 86 = 19$ | $h_{10} < C_{10}$
- State11: $h_{11} = 6$, $C_{11} = 105 – 96 = 9$ | $h_{11} < C_{11}$
- State12: $h_{12} = 7$, $C_{12} = 105 – 94 = 11$ | $h_{12} < C_{12}$
- State13: $h_{13} = 4$, $C_{13} = 105 – 98 = 7$ | $h_{13} < C_{13}$
- State14: $h_{14} = 5$, $C_{14} = 105 – 99 = 6$ | $h_{14} < C_{14}$
- State15: $h_{15} = 3$, $C_{15} = 105 – 101 = 4$ | $h_{15} < C_{15}$
- State16: $h_{16} = 3$, $C_{16} = 105 – 102 = 3$ | $h_{16} < C_{16}$
- FinalState: $h_{17} = 0$, $C_{17} = 105 – 105 = 0$ | $h_{17} <= C_{17}$

- Αρχείο εισόδου: input_04.txt ***[Αρχείο για το οποίο δεν υπάρχει λύση εντός του δοσμένου χρονικού ορίου]***
- Εύρος Χρόνου Αναζήτησης: 0.02 δευτερόλεπτα
- Όριο Χρόνου: 28
- Πλήθος ατόμων: 4
- Χρόνοι Διάσχισης κάθε μέλους της οικογένειας:
  - Boy 3
  - Teen 5
  - Woman 8
  - Grandma 12
- Κόστος (βέλτιστης) λύσης που βρίσκει ο αλγόριθμος: **ΔΕΝ ΒΡΙΣΚΕΙ !**