Τεχνητή Νοημοσύνη 2020-2021
1η ΕΡΓΑΣΙΑ
Δουραχαλής Φίλιππος, 3170045
Νικολάου Ελένη, 3170121

Στα πλαίσια του μαθήματος της Τεχνητής Νοημοσύνης και για την εκπόνηση της πρώτης εργασίας, επιλέξαμε να ασχοληθούμε με το πρώτο πρόβλημα, τη **Διάσχιση της γέφυρας**.

→ Τρόπος Χρήσης & Δυνατότητες του Προγράμματος

Ο χρήστης έχει τη δυνατότητα να τρέξει το πρόγραμμα δίνοντας εξαρχής κάποια ορίσματα στη γραμμή εντολών αλλά και χωρίς ορίσματα, αλληλεπιδρώντας με τη διεπαφή που έχουμε κατασκευάσει.

Με ορίσματα από την γραμμή εντολών (1η λειτουργία): Η εφαρμογή συμπεριφέρεται ανάλογα με το πλήθος των arguments που λαμβάνει στην είσοδο. Αν ο χρήστης δώσει από **2 ορίσματα και πάνω**, δέχεται με τη σειρά τους **χρόνους** (ως ακέραιους θετικούς αριθμούς) που χρειάζεται κάθε μέλος της οικογένειας για να διασχίσει μόνο του τη γέφυρα - κάθε αριθμός/χρόνος στην κονσόλα χωρίζεται με ένα κενό. Ως τελευταίο όρισμα, ζητείται ο συνολικός **χρόνος διάρκειας της λάμπας** (επίσης ως θετικός ακέραιος), δηλαδή ο μέγιστος χρόνος στον οποίο θέλουμε να έχουν περάσει όλα τα μέλη στην αριστερή όχθη. Αν δεν θέλουμε να θέσουμε ανώτατο όριο για το χρόνο της λάμπας, δίνουμε ως τελευταίο όρισμα την τιμή *-1*. ΠΡΟΣΟΧΗ! Ο μη προσδιορισμός μέγιστου χρόνου μεταφράζεται πάντα με την τιμή -1, αν ο χρήστης επιλέξει αντί για -1 να μην δώσει κάποια τιμή, το πρόγραμμα θα θεωρήσει μόνο του το τελευταίο όρισμα ως μέγιστο χρόνο και δεν θα έχει τα επιθυμητά αποτελέσματα.
Αν ο χρήστης δώσει **1 όρισμα**, τότε το πρόγραμμά μας θα μεταφράσει το συγκεκριμένο όρισμα ως το πλήθος των μελών της οικογένειας N. Για δεδομένο πλήθος ατόμων, θα παράξει, μόνο του, **Ν τυχαίους χρόνους** που θα αντιστοιχούν σε κάθε άτομο. Στη συγκεκριμένη λειτουργία δεν ορίζουμε ανώτατο όριο και υπάρχει μέγιστο πλήθος ατόμων που μπορεί να δηλώσει ο χρήστης (σε αντίθεση με όλες τις άλλες λειτουργίες). ΠΡΟΣΟΧΗ! Να μην δοθεί όρισμα μεγαλύτερο του 70. Αυτό συμβαίνει καθώς έχει δηλωθεί στον random κατασκευαστή της *State* ένα ανώτατο όριο για την παραγωγή τυχαίων αριθμών – στην περίπτωσή μας 70. Αυτό θέλει προσοχή καθώς αν δοθεί N>70 τότε δεν θα υπάρχουν μοναδικές τιμές για χρόνους και η εφαρμογή δεν θα τρέξει. Αν βέβαια επιθυμεί κάποιος να το τρέξει με μεγαλύτερο N στη συγκεκριμένη λειτουργία μπορεί να επέμβει και να το αλλάξει στη μεταβλητή `maxRandom` του τυχαίου κατασκευαστή της *State*.

Χωρίς ορίσματα από τη γραμμή εντολών (2η λειτουργία): Η εφαρμογή είναι σε θέση, αν δεν έχει λάβει κάποια αρχικά ορίσματα, να το διαχειριστεί λαμβάνοντας τα δεδομένα που χρειάζεται από συγκεκριμένες ερωτήσεις στον χρήστη. Συγκεκριμένα, με το που τρέξουμε το πρόγραμμα εμφανίζεται το μήνυμα:

```
No input arguments were given. Would you like to enter them now?
[Y] [N]
```

ο χρήστης εδώ επιλέγει αν θέλει να δώσει ορίσματα [Y] και αν δεν θέλει [N]. ΠΡΟΣΟΧΗ! Τα [Y], [N] είναι λατινικοί χαρακτήρες. Σημειώνεται ότι αν επιλέξει [N] τότε το πρόγραμμα τερματίζει.

Εφόσον επιλέξει ότι θέλει να δώσει τα ορίσματα, ακολουθεί το μήνυμα:

```
Specify a time limit for every person to cross?
[Y] [N]
```

με το οποίο ρωτάει αν θέλει ο χρήστης να συγκεκριμενοποιήσει το ανώτατο όριο χρόνου για να περάσουν τα άτομα απέναντι, ή αλλιώς τη διάρκεια της λάμπας (`maxTime`). Αν επιλέξει [Y] εμφανίζεται το μήνυμα

```
Enter the time limit:
```

κάτω απ' το οποίο θα εισάγει το ανώτατο όριο. Στη συνέχεια ή αν έχει επιλέξει [N] για το όριο, καλείται απευθείας να δώσει τους χρόνους που επιθυμεί έναν έναν - για να σταματήσει την εισαγωγή τιμών αρκεί να δώσει την τιμή 0.

```
Enter each person's time to cross the bridge in order. Enter 0 to stop
1)
```

Το πρόγραμμα που έχουμε δημιουργήσει είναι σε θέση, για οποιοδήποτε πλήθος N αρχικών ορισμάτων και δεδομένης τιμής μέγιστου χρόνου `maxTime` (προαιρετικά), να δίνει τη βέλτιστη σειρά με την οποία τα μέλη της οικογένειας μπορούν να διασχίσουν τη γέφυρα. Αρχικά, βρίσκει αν για το συγκεκριμένο χρονικό διάστημα είναι όντως **εφικτό** τα άτομα να διασχίσουν τη γέφυρα. Αν όχι, τότε το πρόγραμμα τερματίζει ενημερώνοντας τον χρήστη ότι ο χρόνος που δόθηκε δεν είναι αρκετός. Αν ο αλγόριθμος έχει καταφέρει σε `maxTime` να βρει κάποια βέλτιστη λύση τότε τυπώνονται με τη σειρά οι μεταβάσεις σε ζεύγη από τη δεξιά όχθη στην αριστερή, αλλά και ενδιάμεσα η μεταφορά της λάμπας πίσω στη δεξιά.

→ Αρχιτεκτονική

Το πρόγραμμα που αναπτύξαμε περιέχει 4 αρχεία κλάσεων.

- Η **κλάση *Person***, περιγράφει αντικείμενα που συμβολίζουν τα άτομα της οικογένειας, το καθένα με 2 πεδία: Ένα υποχρεωτικό πεδίο `int duration`, που περιέχει το χρόνο που χρειάζεται το μέλος της οικογένειας για να διασχίσει τη γέφυρα (μόνο του). Υλοποιεί επίσης το Comparable interface, καθώς θεωρήσαμε ότι χρειάζεται να οριστεί η μέθοδος και τα κριτήρια σύγκρισης μεταξύ των αντικειμένων για να διευκολυνθούμε σε μελλοντικές συγκρίσεις σε άλλες κλάσεις.
- Η **κλάση *State***, περιγράφει τις «καταστάσεις» που δημιουργεί ο αλγόριθμος για να τις αξιολογήσει και να καταλήξει σε βέλτιστη λύση. Ως **αρχική** κατάσταση ορίζουμε το στιγμιότυπο του προβλήματος στο οποίο βρίσκονται όλα τα μέλη που έχει προσδιορίσει ο χρήστης (N άτομα) στην δεξιά όχθη, ενώ **τελική** το στιγμιότυπο που βρίσκονται όλοι στα αριστερά. **Κατάσταση** θεωρούμε ένα στιγμιότυπο του προβλήματος το οποίο έχει «τραβηχτεί» μετά από 2 περάσματα στη γέφυρα: το πρώτο στο οποίο περνάει το ζεύγος με τη λάμπα και το δεύτερο που γυρνάει ένα άτομο από τα αριστερά πίσω. Τα πεδία της είναι:
  - `int N:` ορίζεται απ' το πλήθος των μελών της οικογένειας που δίνονται απ' τον χρήστη,
  - `int timeUpperBound`: το μέγιστο χρονικό όριο που δίνεται απ' τον χρήστη,
  - `List<Person> leftShore`: αντιπροσωπεύει την αριστερή όχθη του προβλήματος, περιέχει τα άτομα που βρίσκονται στη συγκεκριμένη κατάσταση αριστερά,

- *List<Person> **rightShore***: αντιπροσωπεύει τη δεξιά όχθη του προβλήματος, περιέχει τα άτομα που βρίσκονται στη συγκεκριμένη κατάσταση δεξιά**,**
- *State **father***: δείχνει στον «πατέρα» της συγκεκριμένης κατάστασης και είναι null όταν η κατάσταση είναι η αρχική**,**
- ***int totalTime***: εκφράζει τον χρόνο που έχει περάσει απ' την έναρξη του προβλήματος, περιέχει το ***totalTime*** του πατέρα συν το χρόνο που χρειάζεται το ζεύγος που επιλέχθηκε για να περάσει απέναντι, συν τον χρόνο που θέλει για να γυρίσει πίσω αυτός που γυρνάει την λάμπα. Αποτελεί μία απ' τις 2 ευρετικές**,**
- ***int score***: εκφράζει το συνολικό σκορ που υπολογίζει για τη συγκεκριμένη κατάσταση η ευρετική συνάρτηση αφού έχει κληθεί,
- **int**[] **crossing**: περιέχει τους χρόνους των ατόμων που περνούν απέναντι στην αριστερή όχθη,
- *Person **returning***: περιέχει το άτομο που έχει επιλέγεται για να γυρίσει πίσω με τη λάμπα.

Περιλαμβάνει, επίσης, τους κατασκευαστές αλλά και τις αντίστοιχες get, set.

Υπάρχουν και οι μέθοδοι:

+ **boolean** `isTerminal()`: η οποία επιστρέφει true αν η συγκεκριμένη κατάσταση είναι τελική, εξετάζοντας αν το πλήθος των ατόμων στην αριστερή όχθη είναι ίσο με το πλήθος των ατόμων που βρίσκονταν αρχικά στη δεξιά (N) ΚΑΙ αν ο χρόνος ***totalTime*** έχει υπερβεί το όριο που μας έχει δοθεί (***timeUpperBound*)**.
+ `ArrayList<State> getChildren()`: η οποία είναι υπεύθυνη για την παραγωγή των παιδιών της παρούσας κατάστασης, τα οποία και επιστρέφει σε λίστα. Κάθε παιδί παράγεται από έναν συνδυασμό ανά 2 ατόμων της δεξιάς όχθης (ζεύγος για να περάσουν απέναντι) και από ένα άτομο για να επιστρέψει ξανά πίσω/δεξιά με τη λάμπα. Άρα μιλάμε για συνδυασμούς $\binom{i}{2}$, όπου i το πλήθος των ατόμων στα δεξιά, συν τις επιλογές που μπορούν να γίνουν για το άτομο από τα αριστερά που θα επιστρέψει.
+ **void** `heuristic()`: αποτελεί την ευρετική συνάρτηση που χρησιμοποιείται για τον υπολογισμό του *score*.
+ **int** `compareTo(State o)` : αποτελεί βοηθητική συνάρτηση η οποία υλοποιεί τον μηχανισμό σύγκρισης μεταξύ 2 αντικειμένων State.
+ **void** `listToPrint(List<State> list)` : δημιουργεί λίστα που περιέχει την κατάσταση αυτή και τους προγόνους της. Καλείται στη Main για την τελική κατάσταση που έχει προκύψει απ' τον αλγόριθμο και προσθέτει σε λίστα τους κόμβους από το μονοπάτι που ακολουθήθηκε.
+ **void** `print()`:εκτυπώνει την παρούσα κατάσταση.

```
1η σειρά εξόδου: ο συνδυασμός ατόμων που επιλέγονται να
διασχίσουν
Left Shore     Right Shore
{_,..,_} <----- {_,...,_)
2η σειρά εξόδου: το άτομο που επιλέγεται γυρνάει πίσω με τη
λάμπα
Left Shore     Right Shore
{_,..,_} -----> {_,...,_)
Elapsed Time = minutes from the start
```

Παράδειγμα από εκτέλεση:

Αρχική κατάσταση

```
{                   } -----> {1, 3, 6, 8, 12 }
Elapsed Time = 0

{1, 3               } <----- {6, 8, 12 }
{3,                 } -----> {1, 6, 8, 12 }
Elapsed Time = 4

{8, 12, 3           } <----- {1, 6 }
{8, 12              } -----> {1, 3, 6 }
Elapsed Time = 19

{1, 3, 8, 12        } <----- {6, }
{8, 12, 3           } -----> {1, 6 }
Elapsed Time = 23

{8, 12, 3, 1, 6     } -----> {}
Elapsed Time = 29
```

Οι χρόνοι των ατόμων που διασχίζουν κάθε φορά τη γέφυρα είναι με **bold.**

- Η **κλάση *AStar***, περιέχει την υλοποίηση του αλγορίθμου τεχνητής νοημοσύνης. Στη συγκεκριμένη εργασία, επιλέξαμε να υλοποιήσουμε τον αλγόριθμο A* με κλειστό σύνολο. Η κλάση έχει 2 πεδία, `List<State>` **`states`** η λίστα που αντιπροσωπεύει το μέτωπο της αναζήτησης και περιέχει όλες τις καταστάσεις προς εξερεύνηση, `HashSet<State>` **`closedSet`** το κλειστό σύνολο στο οποίο βρίσκονται όλες οι καταστάσεις που έχουν ήδη εξερευνηθεί.
- Η **κλάση Main**, αποτελεί την εφαρμογή για τον χρήστη. Όπως περιγράψαμε και στον τρόπο χρήσης έχει 2 λειτουργίες είτε να παίρνει είσοδο απ' την κονσόλα είτε να κάνει η ίδια τις αντίστοιχες ερωτήσεις στον χρήστη.
  Η main με τα ορίσματα που δέχεται απ' το χρήστη κατασκευάζει την αρχική κατάσταση `initialState` και τη δίνει στον αλγόριθμο A* για να παράξει το βέλτιστο μονοπάτι για την τελική `terminalState`. Στη συνέχεια, με τη χρήση της μεθόδου `listToPrint` δημιουργείται λίστα για τους προγόνους της `terminalState.` Με έναν Iterator τη διατρέχουμε και εκτυπώνουμε τις καταστάσεις προγόνους με την `print().`
  Επίσης, με τις μεταβλητές **`long`** `start` και **`long`** `finish,` πριν και μετά την εκτέλεση του αλγορίθμου, μετράμε το χρόνο που χρειάστηκε το μηχάνημά μας για να τον εκτελέσει.

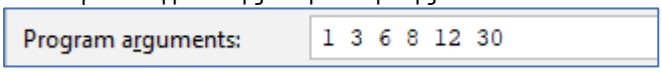
→ <u>Μέθοδος Τεχνητής Νοημοσύνης που χρησιμοποιήθηκε</u>
Επιλέξαμε να υλοποιήσουμε τη λύση του προβλήματος με τη χρήση του αλγορίθμου **A***. Αυτό διότι στο πρόβλημά μας δεν ζητείται απλά να βρεθεί μία λύση - ένα μονοπάτι που θα καταλήξει σε τελική κατάσταση, αλλά μία **βέλτιστη** λύση - ένα βέλτιστο μονοπάτι. Ο στόχος είναι να μπορούμε να

κρίνουμε για δεδομένους χρόνους αν είναι εφικτό όλα τα άτομα να περάσουν απέναντι σε ένα δοσμένο χρονικό διάστημα. Γνωρίζουμε πως ο αλγόριθμός μας θα παράγει πάντα ως τελική κατάσταση την κατάσταση που προέκυψε απ' το βέλτιστο μονοπάτι, άρα και τον βέλτιστο χρόνο για τα συγκεκριμένα δεδομένα. Έτσι θα είμαστε σίγουροι για το αν το πρόβλημα έχει λύση για το συγκεκριμένο χρονικό όριο που μας δόθηκε. Επομένως ο A* είναι κατάλληλος για την περίπτωσή μας.

Για να επιτύχουμε βέλτιστη λύση με τον A* πρέπει να χρησιμοποιήσουμε μία αντίστοιχη ευρετική συνάρτηση. Ο A* χρειάζεται μία ευρετική της μορφής f(n)=h(n)+g(n), όπου h(n): το κόστος της κατάστασης που αντιστοιχεί στον κόμβο *n* και g(n): το κόστος από τη ρίζα έως τον *n*. Στο συγκεκριμένο πρόβλημα επιλέξαμε να θεωρήσουμε τη συνάρτηση **h(n) ως τον χρόνο που απομένει αν περνούσαν ένα-ένα τα άτομα της δεξιάς όχθης αριστερά**, αθροίζοντας δηλαδή τους χρόνους των αντικειμένων Person της rightShore λίστας. Αντίστοιχα, θεωρήσαμε τη συνάρτηση **g(n) ως τον χρόνο που έχει περάσει απ' την αρχική κατάσταση έως τον κόμβο n - την παρούσα κατάσταση**.
Η ευρετική που προκύπτει f(n), ως το άθροισμα αυτών των δύο και αποθηκεύεται ως αποτέλεσμα στο πεδίο `score` της κάθε κατάστασης, είναι αποδεκτή καθώς υποεκτιμά κάθε φορά τα βήματα προς τη λύση.

→ <u>Παραδείγματα Εκτέλεσης</u>

1) Το παράδειγμα της εκφώνησης:



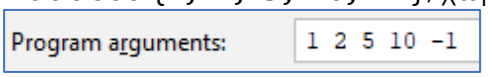
με αρχικά ορίσματα `{1,3,6,8,12}` και `maxTime=30`

Έχουμε την παρακάτω έξοδο,

```
{                   } -----> {1, 3, 6, 8, 12 }
Elapsed Time = 0

{1, 3               } <----- {6, 8, 12 }
{3,                 } -----> {1, 6, 8, 12 }
Elapsed Time = 4

{8, 12, 3           } <----- {1, 6 }
{8, 12              } -----> {1, 3, 6 }
Elapsed Time = 19

{1, 3, 8, 12        } <----- {6, }
{8, 12, 3           } -----> {1, 6 }
Elapsed Time = 23

{8, 12, 3, 1, 6     } -----> {}
Elapsed Time = 29
```

που εκτελέστηκε σε χρόνο 0.003 sec.

2) Με είσοδο `{1, 2, 5, 10, -1}`, χωρίς `maxTime`.

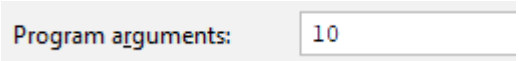


Παράγεται η έξοδος:

```
{              } -----> {1, 2, 5, 10 }
Elapsed Time = 0

{1, 2          } <----- {5, 10 }
{2,            } -----> {1, 5, 10 }
Elapsed Time = 3

{5, 10, 2      } <----- {1, }
{5, 10         } -----> {1, 2 }
Elapsed Time = 15

{5, 10, 1, 2   } -----> {}
Elapsed Time = 17
```

Και έχουμε χρόνο εκτέλεσης 0.001 sec.

3) Για είσοδο:



Παράγεται η έξοδος (για τυχαίους 10 χρόνους):

```
{                                  } -----> {3, 68, 32, 56, 26, 69, 5, 42, 38, 54 }
Elapsed Time = 0

{69, 5                             } <----- {3, 26, 32, 38, 42, 54, 56, 68 }
{69                                } -----> {3, 5, 26, 32, 38, 42, 54, 56, 68 }
Elapsed Time = 10

{3, 5, 69                          } <----- {26, 32, 38, 42, 54, 56, 68 }
{69, 5                             } -----> {3, 26, 32, 38, 42, 54, 56, 68 }
Elapsed Time = 18

{56, 68, 69                        } <----- {3, 26, 32, 38, 42, 54 }
{69, 56, 68                        } -----> {3, 5, 26, 32, 38, 42, 54 }
Elapsed Time = 91

{3, 5, 69, 56, 68                  } <----- {26, 32, 38, 42, 54 }
{69, 56, 68, 5                     } -----> {3, 26, 32, 38, 42, 54 }
Elapsed Time = 99

{42, 54, 69, 56, 68                } <----- {3, 26, 32, 38 }
{69, 56, 68, 42, 54                } -----> {3, 5, 26, 32, 38 }
Elapsed Time = 158
```

```
{3, 5, 69, 56, 68, 42, 54              } <----- {26, 32, 38 }
{69, 56, 68, 42, 54, 5                 } -----> {3, 26, 32, 38 }
Elapsed Time = 166

{32, 38, 69, 56, 68, 42, 54            } <----- {3, 26 }
{69, 56, 68, 42, 54, 32, 38            } -----> {3, 5, 26 }
Elapsed Time = 209

{3, 5, 69, 56, 68, 42, 54, 32, 38      } <----- {26 }
{69, 56, 68, 42, 54, 32, 38, 5         } -----> {3, 26 }
Elapsed Time = 217

{69, 56, 68, 42, 54, 32, 38, 5, 3, 26  } -----> {}
Elapsed Time = 243

The algorithm run in: 0.022 sec
```

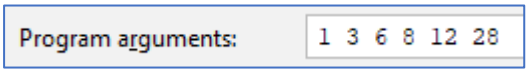
που εκτελείται σε 0.022 sec.

4) Για (ακραία) είσοδο: 70, εκτελείται σε 15,684 sec

```
{1, 2, 70, 68, 69, 66, 67, 64, 65, 62, 63, 60, 61, 58, 59, 56, 57, 54, 55, 52, 53, 50, 51, 48, 49, 46, 47, 44, 45, 42, 43, 40, 41, 38, 39,
{70, 68, 69, 66, 67, 64, 65, 62, 63, 60, 61, 58, 59, 56, 57, 54, 55, 52, 53, 50, 51, 48, 49, 46, 47, 44, 45, 42, 43, 40, 41, 38, 39, 36, 37
Elapsed Time = 1369

{6, 7, 70, 68, 69, 66, 67, 64, 65, 62, 63, 60, 61, 58, 59, 56, 57, 54, 55, 52, 53, 50, 51, 48, 49, 46, 47, 44, 45, 42, 43, 40, 41, 38, 39,
{70, 68, 69, 66, 67, 64, 65, 62, 63, 60, 61, 58, 59, 56, 57, 54, 55, 52, 53, 50, 51, 48, 49, 46, 47, 44, 45, 42, 43, 40, 41, 38, 39, 36, 37
Elapsed Time = 1378

{1, 2, 70, 68, 69, 66, 67, 64, 65, 62, 63, 60, 61, 58, 59, 56, 57, 54, 55, 52, 53, 50, 51, 48, 49, 46, 47, 44, 45, 42, 43, 40, 41, 38, 39,
{70, 68, 69, 66, 67, 64, 65, 62, 63, 60, 61, 58, 59, 56, 57, 54, 55, 52, 53, 50, 51, 48, 49, 46, 47, 44, 45, 42, 43, 40, 41, 38, 39, 36, 37
Elapsed Time = 1381

{4, 5, 70, 68, 69, 66, 67, 64, 65, 62, 63, 60, 61, 58, 59, 56, 57, 54, 55, 52, 53, 50, 51, 48, 49, 46, 47, 44, 45, 42, 43, 40, 41, 38, 39,
{70, 68, 69, 66, 67, 64, 65, 62, 63, 60, 61, 58, 59, 56, 57, 54, 55, 52, 53, 50, 51, 48, 49, 46, 47, 44, 45, 42, 43, 40, 41, 38, 39, 36, 37
Elapsed Time = 1388

{1, 2, 70, 68, 69, 66, 67, 64, 65, 62, 63, 60, 61, 58, 59, 56, 57, 54, 55, 52, 53, 50, 51, 48, 49, 46, 47, 44, 45, 42, 43, 40, 41, 38, 39,
{70, 68, 69, 66, 67, 64, 65, 62, 63, 60, 61, 58, 59, 56, 57, 54, 55, 52, 53, 50, 51, 48, 49, 46, 47, 44, 45, 42, 43, 40, 41, 38, 39, 36, 37
Elapsed Time = 1391

{70, 68, 69, 66, 67, 64, 65, 62, 63, 60, 61, 58, 59, 56, 57, 54, 55, 52, 53, 50, 51, 48, 49, 46, 47, 44, 45, 42, 43, 40, 41, 38, 39, 36, 37
Elapsed Time = 1394

The algorithm run in: 15.684 msec
```

Τα παραπάνω εκτελέστηκαν σε μηχάνημα με επεξεργαστή i5-4200M και 8GB RAM.

5) Με είσοδο



Δεν υπάρχει λύση για τα συγκεκριμένα δεδομένα και το πρόγραμμα επιστρέφει:

```
Couldn't find solution
The algorithm run in: 0.051 sec
```

6) Για εκτέλεση χωρίς args

```
No input arguments were given. Would you like to enter them now?
[Y] [N]
Y
Specify a time limit for every person to cross?
[Y] [N]
N
Enter each person's time to cross the bridge in order. Enter 0 to stop
1) 2

2) 8

3) 12

4) 20

5) 25

6) 0
```

Έχουμε την έξοδο:

```
{                   } -----> {2, 8, 12, 20, 25 }
Elapsed Time = 0

{2, 8               } <----- {12, 20, 25 }
{8,                 } -----> {2, 12, 20, 25 }
Elapsed Time = 10

{20, 25, 8          } <----- {2, 12 }
{20, 25             } -----> {2, 8, 12 }
Elapsed Time = 43

{2, 8, 20, 25       } <----- {12 }
{20, 25, 8          } -----> {2, 12 }
Elapsed Time = 53

{20, 25, 8, 2, 12   } -----> {}
Elapsed Time = 65

The algorithm run in: 0.0 sec
```