1. ΕΥΡΕΤΙΚΗ ΣΥΝΑΡΤΗΣΗ

1.1 Πρόβλημα

Στην εργασία αυτή επιλέξαμε να επιλύσουμε το πρόβλημα διάσχισης ποταμού.

1.2 Ευρετική Συνάρτηση

Έστω αρχικά ότι βρισκόμαστε σε μία κατάσταση s και έστω ότι θέλουμε να υπολογίσουμε το κόστος από την s μέχρι την βέλτιστη λύση.

Έστω z_s το άθροισμα των χρόνων που απαιτείται για να διασχίσουνε διαδοχικά τα ζεύγη ατόμων, τα οποία κατασκευάζονται με τον εξής τρόπο: (1) επιλέγουμε το πιο αργό άτομο το οποίο δεν είναι μέρος κάποιου ζεύγους και το βάζουμε σε ζεύγος με το αμέσως ταχύτερο. Αν ΔΕΝ έχει απομείνει δεύτερο άτομο, το βάζουμε σε «ζεύγος» μόνο του. (2) Αν έχει απομείνει κάποιο άτομο το οποίο δεν είναι σε ζεύγος, επιστρέφουμε στο βήμα (1). Για παράδειγμα αν τα άτομα στη δεξιά όχθη στην κατάσταση s έχουν χρόνους 1, 3, 6, 8 και 12 αντιστοίχως, τότε τα ζεύγη θα είναι {12, 8}, {6, 3}, {1}, συνεπώς $z_s = \max(\{12,8\}) + \max(\{6,3\}) + 1 = 12 + 6 + 1 = 19$. Αν τα άτομα δεξιά είχαν χρόνους 3, 10, 5, 20, 2, 1 τότε τα ζεύγη θα ήταν {20, 10}, {5, 3}, {2, 1}, άρα $z_s = \max(\{20,10\}) + \max(\{5,3\}) + \max(\{2,1\}) = 20 + 5 + 2 = 27$.

Έστω N_s ο αριθμός των φορών που πρέπει να επιστρέψει κάποιος από τα αριστερά στα δεξιά με την λάμπα, προκειμένου μετέπειτα να διασχίσει κάποιο ζεύγος τη γέφυρα, ως ότου να φτάσουμε στην λύση, έχοντας ξεκινήσει από την s. Το N_s δίδεται από την παρακάτω συνάρτηση:

$$N_s = \begin{cases} |R|, & \text{αν η λάμπα είναι στα αριστερά} \\ |R| - 2, & \text{αν η λάμπα είναι στα δεξιά} \end{cases}$$

όπου $|R|$ είναι το πλήθος των ατόμων που έχουν απομείνει στα δεξιά.

Απόδειξη

Προτού αποδείξουμε ότι η ευρετική είναι αποδεκτή, πρέπει να αποδείξουμε ότι το N_s πράγματι καθορίζεται από την δοθείσα συνάρτηση. Θα αποδείξουμε τον κάθε κλάδο της συνάρτησης ξεχωριστά.

Έστω λοιπόν ότι η λάμπα είναι στα αριστερά και έχουμε $|R|$ άτομα στη δεξιά μεριά. Προφανώς προκειμένου να περάσει ένα άτομο από τα δεξιά στα αριστερά, πρέπει να επιστρέψει κάποιος από τα αριστερά με τη λάμπα και έπειτα να διασχίσει την γέφυρα από τα δεξιά στα αριστερά με κάποιον άλλον. Συνεπώς για $|R|$ άτομα απαιτείται να διασχιστεί η γέφυρα από τα αριστερά προς τα δεξιά $|R|$ φορές, άρα $N_S = |R|$.

Έστω τώρα ότι η λάμπα είναι στα δεξιά και ότι βρίσκονται $|R|$ άτομα στα δεξιά. Το επόμενο βήμα σε κάθε λύση είναι να περάσουν δύο άτομα από τα δεξιά στα αριστερά. Αφού περάσουν τα δύο άτομα, πλέον υπάρχουν $|R| - 2$ άτομα στα δεξιά και η λάμπα βρίσκεται στα αριστερά, συνεπώς χρησιμοποιώντας το προηγούμενο σκέλος της απόδειξης προκύπτει ότι $N_s = |R| - 2$. Ο.Ε.Δ.

Έστω t_{min} ο ελάχιστος χρόνος που απαιτείται για να διασχίσει κάποιο άτομο τη γέφυρα, ανεξαρτήτως σε ποια μεριά στέκεται. Για παράδειγμα για τους χρόνους 1, 3, 6, 8 και 12 έχουμε ότι $t_{min} = 1$.

Η ευρετική συνάρτηση που χρησιμοποιήσαμε λοιπόν είναι η $h(n) = z_s + N_s \cdot t_{min}$

1.3 Απόδειξη αποδεκτής ευρετικής

Μπορούμε να διαχωρίσουμε την επίλυση του προβλήματος σε δύο μέρη: (i) Την μεταφορά των ατόμων από δεξιά στα αριστερά και (ii) την επιστροφή κάποιου ατόμου από τα αριστερά προς τα δεξιά με την λάμπα, όσες φορές αυτό χρειαστεί από την αρχή μέχρι το τέλος της λύσης.

Έστω ότι βρισκόμαστε σε μία κατάσταση s. Ο πιο αποδοτικός τρόπος να διασχίσουνε τα άτομα τη γέφυρα, αφαιρώντας τον περιορισμό ότι χρειάζονται τη λάμπα για να περάσουνε, είναι να τα διατάξουμε σε ζεύγη με τον τρόπο που περιγράψαμε στο **1.2**.

Απόδειξη
Προφανώς για 0 ή 1 ζεύγη δεν υπάρχει πιο αποδοτικός τρόπος. Έστω ότι για παραπάνω από ένα ζεύγος υπάρχει κάποιος πιο αποδοτικός τρόπος. Αυτό σημαίνει ότι υπάρχουν 2 ζεύγη, έστω $\mathrm{A} = \{\mathrm{x}, \mathrm{y}\}$ και $\mathrm{B} = \{\mathrm{z}, \mathrm{w}\}$, όπου $\mathrm{x} > \mathrm{y} > \mathrm{z} > \mathrm{w}$ και $\mathrm{C} = \max(\mathrm{A}) + \max(\mathrm{B}) = \mathrm{x} + \mathrm{z}$ το κόστος αυτών των δύο ζευγών, των οποίων αν εναλλάξουμε (swap) κάποιο μέλος του ενός με κάποιο μέλος του άλλου, θα λάβουμε μία πιο αποδοτική λύση. Έστω ότι εναλλάσσουμε το x με κάποιο από τα μέλη του B, είτε το z είτε το w. Τότε $\max(\mathrm{A}) + \max(\mathrm{B}) = \mathrm{y} + \mathrm{x} > \mathrm{C}$, συνεπώς το κόστος θα είναι μεγαλύτερο. Αν εναλλάξουμε το y με κάποιο από τα w ή z τότε προκύπτει το ίδιο αποτέλεσμα, συνεπώς αν αλλάξουμε τα ζεύγη το κόστος θα είναι πάντα μεγαλύτερο. Ο.Ε.Δ

Χρησιμοποιώντας τους ορισμούς του **1.2**, προκύπτει ότι $z_s \leq$ πραγματικό κόστος διάσχισης ατόμων από την κατάσταση s, δηλαδή το κόστος του (i).

Ας ασχοληθούμε τώρα με το (ii). Προφανώς ο βέλτιστος τρόπος να επιστρέφει κάποιος από τα αριστερά στα δεξιά (σε έναν ιδανικό κόσμο) είναι αυτός ο «κάποιος» να είναι το πιο γρήγορο άτομο από όλα. Συνεπώς, χρησιμοποιώντας τους ορισμούς του **1.2**, έπεται ότι $N_S \cdot t_{min} \leq$ κόστος του (ii).

Συνεπώς $z_S + N_S \cdot t_{min} \leq$ κόστος (i) + κόστος (ii) $\leftrightarrow h(n) \leq C^*(n)$. Ο.Ε.Δ

Δεν θα ελέγξουμε αν η συνάρτηση είναι συνεπής, καθώς πειραματικά διαπιστώσαμε ότι με τη χρήση κλειστού συνόλου η εύρεση της λύσης είναι πιο αργή, οπότε είτε δεν είναι συνεπής, είτε η χρήση κλειστού συνόλου καθυστερεί την εκτέλεση περισσότερο από ότι την επιταχύνει.

2. ΔΟΜΗ ΚΩΔΙΚΑ

2.1 Οντότητες και αρχεία

Οι κύριες κλάσεις του προγράμματος είναι **Person, State, Frontier** και **SpaceSearcher** και για την κάθε μία υπάρχει ένα αρχείο **.h** και ένα αρχείο **.cpp**. Επίσης υπάρχει το αρχείο **main.cpp** που περιέχει το entry point του προγράμματος, καθώς επίσης υπάρχει και το **Utility.cpp και το Utility.h**.

2.1.1 main

Η main ξεκινάει κάνοντας έλεγχο στα command line arguments, εμφανίζοντας τα κατάλληλα μηνύματα σε περίπτωση που υπάρχει κάποιο σφάλμα. Αφού κάνει parse το N και το max_time, δημιουργεί N άτομα με τους δοσμένους χρόνους, δημιουργεί το αρχικό state, εκτελεί τον αλγόριθμο της αναζήτησης με περιορισμό max_time και έπειτα αν βρεθεί λύση, την εκτυπώνει.

2.1.2 Person

Η κλάση αυτή αναπαριστά ένα άτομο το οποίο θα διασχίσει τη γέφυρα. Συγκρατεί μόνο δύο πληροφορίες: το **id** του κάθε ατόμου το οποίο το διακρίνει και τον **χρόνο** όπου χρειάζεται για να διασχίσει τη γέφυρα. Το id του κάθε ατόμου είναι ένας διαφορετικός πρώτος αριθμός.

2.1.3 State

Η κλάση State αναπαριστά μία κατάσταση του προβλήματος και περιέχει πληροφορίες για αυτή, καθώς και συναρτήσεις μετάβασης και υπολογισμού κόστους. Περιέχει **2 set** τα οποία αναπαριστούν την αριστερή και δεξιά όχθη του ποταμού και περιέχουν τα άτομα που βρίσκονται στην κάθε μία αντιστοίχως. Περιέχει μία αναφορά στην **parent** κατάσταση, την οποία θα χρησιμοποιήσουμε στο τέλος έτσι ώστε να εκτυπώσουμε τη λύση. Επίσης περιέχει ένα **id** το οποίο χρησιμοποιείται για να ξεχωρίζουμε τις καταστάσεις μεταξύ τους και υπολογίζεται ως εξής:

$$id = \left(\prod_{x \in S} ID(x)\right) + C$$

Όπου S το σύνολο της όχθης με τα λιγότερα άτομα, $ID(X)$ είναι το id του ατόμου x και $C = 1$ αν η λάμπα είναι στα δεξιά, ειδάλλως $C = 0$. Με αυτόν τον τρόπο είναι σίγουρο ότι κάθε id κατάστασης είναι μοναδικό, επειδή κάθε άτομο έχει ως id έναν διαφορετικό πρώτο αριθμό (Θεμελιώδες θεώρημα αριθμητικής). Η κλάση αυτή επίσης διατηρεί κάποιες πληροφορίες σχετικά με το κόστος της κατάστασης, οι οποίες μπορούν να χρησιμοποιηθούν στην αναζήτηση λύσης.

Η κλάση αυτή είναι επίσης υπεύθυνη για την παραγωγή child καταστάσεων χρησιμοποιώντας τις κατάλληλες μεταβάσεις, καθώς και για τον υπολογισμό του $A^*\ score$ χρησιμοποιώντας **την** ευρετική που περιγράψαμε στο **1.2**.

2.1.4 Frontier

Η κλάση Frontier αναπαριστά το μέτωπο της αναζήτησης και υλοποιείται με τη χρήση priority queue, έτσι ώστε κάθε φορά που εισάγεται μία κατάσταση στο frontier, να χρησιμοποιείται το $A^*\ score$ της κατάστασης αυτομάτως ώστε αυτές με το χαμηλότερο να μπαίνουν μπροστά στην ουρά. Έτσι στην αναζήτηση βρίσκουμε γρήγορα την κατάσταση με το πιο χαμηλό $A^*\ score$.

2.1.5 SpaceSearcher

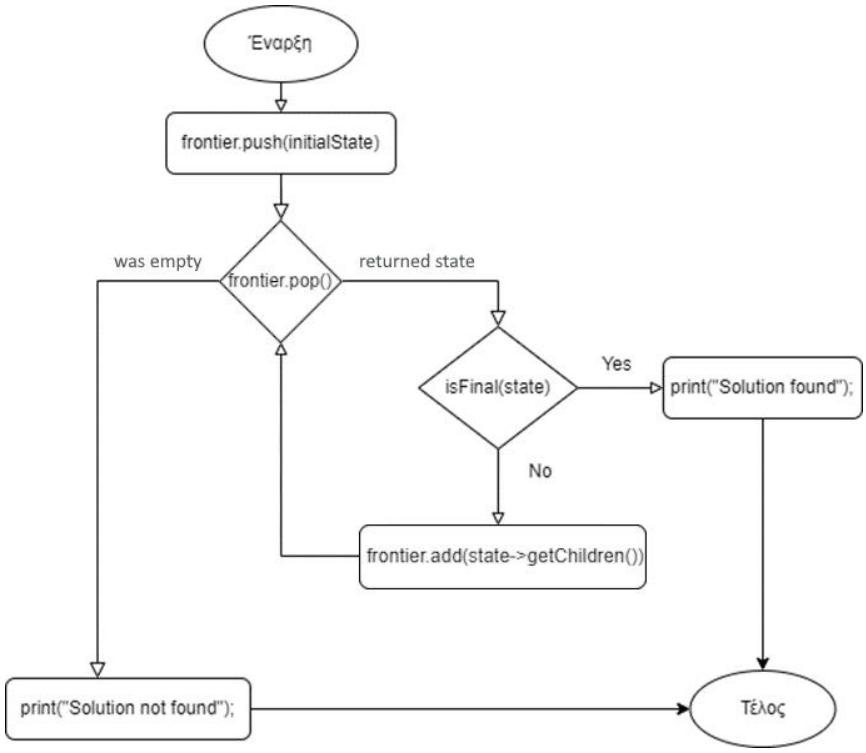
Η κλάση αυτή υλοποιεί την αναζήτηση A^* χρησιμοποιώντας τις 3 προηγούμενες κλάσεις και επιστρέφει τις καταστάσεις που οδηγούν στη λύση με τη σειρά.

2.1.6 Utility.cpp

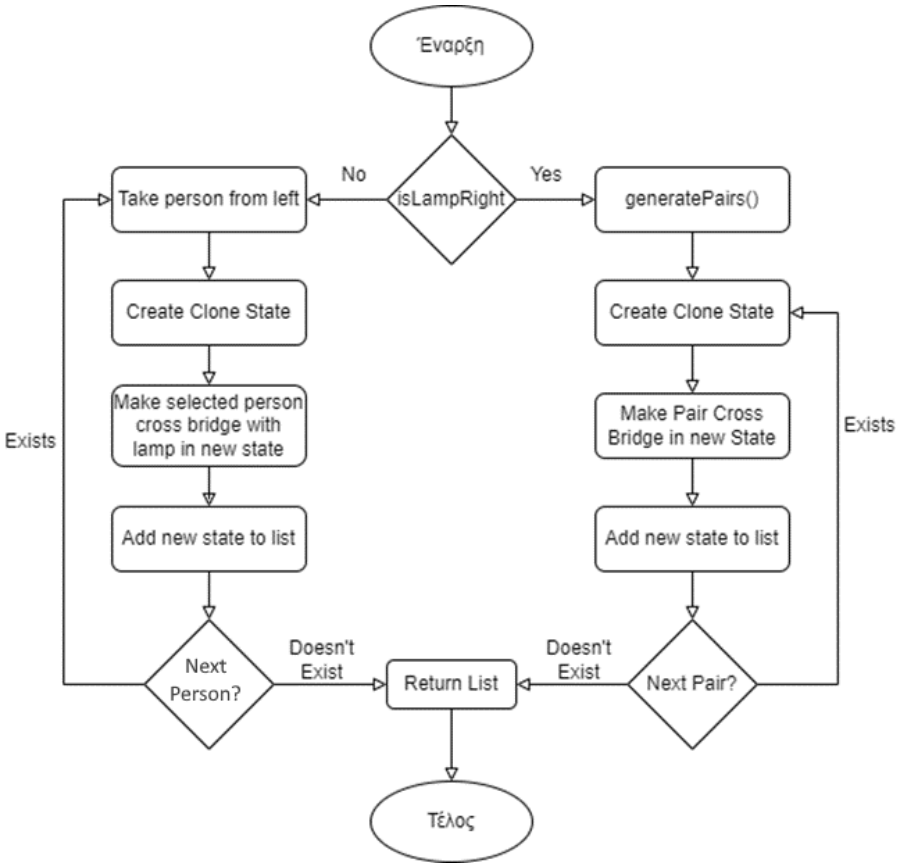
Το αρχείο αυτό υλοποιεί 2 συναρτήσεις: μία συνάρτηση που ελέγχει αν ένας αριθμός είναι πρώτος (για το ID του Person) και μία συνάρτηση που παράγει όλα τα ζεύγη 2 ατόμων από ένα multiset (για την παραγωγή child καταστάσεων).

2.2 Διαγράμματα Ροής

Το διάγραμμα ροής της αναζήτησης A^* παρουσιάζεται παρακάτω (λίγο απλοποιημένα):



Η getChildren() κατασκευάζει μία λίστα με child καταστάσεις και την επιστρέφει. Το διάγραμμα ροής της ρουτίνας παρουσιάζεται παρακάτω:



3. ΔΟΚΙΜΕΣ

3.1 Command Line Arguments

Η εκτέλεση του προγράμματος από το command line πρέπει να έχει την εξής μορφή:

"Bridge Crossing.exe" <N> <T> <T_1, T_2, …, T_N>

Όπου N = ο αριθμός των ατόμων, T = μέγιστο κόστος λύσης, T_i = ο χρόνος που χρειάζεται για να διασχίσει το i-οστό άτομο τη γέφυρα.

3.2 Δοκιμές με διαφορετικές παραμέτρους

Παρακάτω θα κάνουμε μερικές δοκιμές με διαφορετικό T έτσι ώστε να επιβεβαιώσουμε ότι η λύση που βρίσκουμε είναι όντως βέλτιστη, καθώς αν T_{opt} είναι το βέλτιστο κόστος της λύσης, τότε αν θέσουμε $T = T_{opt} - 1$ το πρόγραμμα θα πρέπει να αποτύχει να βρει λύση

Οι πράσινες τιμές είναι το άνω όριο κόστους που δοκιμάσαμε πειραματικά προκειμένου να βρούμε την βέλτιστη λύση, ενώ οι κόκκινες είναι ένα λιγότερο από το βέλτιστο, ώστε να αναδείξουμε ότι το πρόγραμμα δεν βρίσκει λύση όταν βάζουμε όριο μικρότερο από το βέλτιστο.

Επιτυχημένη Δοκιμή	Αποτυχημένη Δοκιμή

>"Bridge Crossing.exe" 5 30 1 3 6 8 12
`Solution Cost: 29`
`Solution found in 0.000000 seconds`
	>"Bridge Crossing.exe" 5 28 1 3 6 8 12
	`No solution found`

>"Bridge Crossing.exe" 8 150 2 5 7 10 13 17 19 24
`Solution Cost: 91`
`Solution found in 0.341000 seconds`
	>"Bridge Crossing.exe" 8 90 2 5 7 10 13 17 19 24
	`No solution found`

>"Bridge Crossing.exe" 10 150 1 3 6 8 12 14 17 19 23 27
`Solution Cost: 99`
`Solution found in 1.684000 seconds`
	>"Bridge Crossing.exe" 10 98 1 3 6 8 12 14 17 19 23 27
	`No solution found`

>"Bridge Crossing.exe" 12 300 1 1 2 3 5 8 13 21 34 55 89 144
`Solution Cost: 247`
`Solution found in 2.515000 seconds`
	>"Bridge Crossing.exe" 12 246 1 1 2 3 5 8 13 21 34 55 89 144
	`No solution found`

Όπως βλέπουμε το πρόγραμμα βρίσκει τη βέλτιστη λύση σε κάθε δοκιμή, και επειδή έχουμε αποδείξει ότι η ευρετική είναι αποδεκτή, είμαστε βέβαιοι ότι αυτό θα συμβαίνει πάντα.