Η εργασία αυτή είναι μία εφαρμογή που προσομοιώνει ένα πλέγμα διαστάσεων 100x100 μέσα στο οποίο κινούνται διάφοροι χρήστες (2 υγιείς και 2 μολυσμένοι από κορωνοϊό) και δίνει τη δυνατότητα στο διαχειριστή της εφαρμογής να ελέγχει αν οι υγιείς χρήστες βρίσκονται σε κίνδυνο να μολυνθούν. Για την υλοποίηση της εργασίας έπρεπε να κατασκευάσουμε τις δικές μας δυναμικές δομές δεδομένων, ώστε το πρόγραμμα να έχει τη μέγιστη αποδοτικότητα. Κομβικό σημείο της εργασίας ήταν η δημιουργία της συνδεδεμένης λίστας-αλυσίδας (LinkedList), η οποία μας επιτρέπει την καταγραφή και την αποθήκευση των δεδομένων κάθε χρήστη. Τα δεδομένα αυτά αποθηκεύονται ανά 30 δευτερόλεπτα σε έναν νέο κόμβο της λίστας. Ο κόμβος δημιουργείται μέσω μίας δομής (struct) μέσα στην οποία έχουμε ορίσει ένα δείκτη next ο οποίος θα δείχνει στη διεύθυνση του επόμενου κόμβου, ένα δείκτη first που θα δείχνει στη διεύθυνση του πρώτου κόμβου, τις συντεταγμένες x, y που θα έχει ο χρήστης μία συγκεκριμένη στιγμή μέσα στο πλέγμα και τη μεταβλητή time που θα περιέχει την ώρα που δημιουργήθηκε ο κόμβος. Στη συνέχεια κατασκευάζουμε την τάξη LinkedList στην οποία περιέχονται οι δείκτες first και last που δείχνουν στη διεύθυνση του πρώτου και του τελευταίου κόμβου αντίστοιχα και αρχικοποιούνται από τον constructor με την τιμή NULL, καθώς επίσης περιέχονται πίνακες για κάθε χρήστη που κρατούν τις διευθύνσεις που θα δείχνουν στο πρώτο κόμβο κάθε νέας ημέρας. Στη συνάρτηση main φτιάχνουμε 4 αντικείμενα τύπου LinkedList, ένα για κάθε χρήστη (2 υγιείς και 2 μολυσμένοι από κορωνοϊό). Σημαντικό είναι να αναφέρουμε ότι έχουμε προσομοιώσει την ώρα, έτσι ώστε με την εκκίνηση του προγράμματος ξεκινάμε από την Day 0, 0 δευτερόλεπτα και μέσω της αύξησης ενός μετρητή i προσομοιώνουμε το πέρασμα 30 δευτερολέπτων. Ο μετρήτης i αρχικοποιείται με 0 στην αρχή της ημέρας και μέσω ένος βρόχου αυξάνεται κατά 1 μέχρι να πάρει την τιμή 2880 (2880*30=86400 που είναι τα δευτερόλεπτα που αντιστοιχούν σε μία ημέρα). Όταν βγει από το βρόχο ο μετρητής i τότε αυξάνουμε το μετρητή j κατά 1 τον οποίο έχουμε επίσης αρχικοποιήσει με 0 και δηλώνει την ημέρα στην οποία βρισκόμαστε. Όλη αυτή η διαδικασία εμπεριέχεται μέσα σε έναν εξωτερικό βρόχο στην αρχή του οποίου ξανά αρχικοποιείται το i με 0. Αυτός ο βρόχος εκτελείται για τουλάχιστον 7 ημέρες και μετά δίνουμε τη δυνατότητα στο διαχειριστή της εφαρμογής αν θέλει να σταματήσει το πρόγραμμα αλλάζοντας τη τιμή της μεταβλητής cont (continue), η οποία είναι αρχικοποιημένη με 1 και χάρη σε αυτή τρέχει ο εξωτερικός βρόχος. Μέσα στον εσωτερικό βρόχο πραγματοποιείται η καταγραφή των δεδομένων κάθε χρήστη μέσα στην λίστα. Αυτό είναι εφικτό μέσω της συνάρτησης add στην οποία δίνουμε τις συντεταγμένες και την ώρα του κάθε χρήστη για κάθε χρόνικη στιγμή ανά 30 δευτερόλεπτα και μεταφέρει αυτά τα δεδομένα στο τελευταίο κάθε φορά κόμβο της λίστας. Οι συντεταγμένες αυτές αρχικοποιούνται με μία τυχαία τιμή εντός των ορίων του πλέγματος και στη συνέχεια μεταβάλλουμε την τιμή αυτή μέσω της συνάρτησης Movement που ακολουθεί το Random Waypoint Model. Ουσιαστικά η Movement είτε διαλέγει μία τυχαία κατεύθυνση προς την οποία θα κινηθεί ο χρήστης είτε δεν αλλάζει τη θέση του. Για να προσεγγίσουμε όσο γίνεται την πραγματικότητα αυξάνουμε την πιθανότητα να μείνει ακίνητος αν είχε μείνει στάσιμος για λίγη ώρα, ενώ αντίστοιχα τη μειώνουμε εάν κινείται. Θεωρούμε ότι αυτό που μας βοήθησε να πετύχουμε το στόχο της εργασίας, δηλαδή τη δημιουργία δομών με χαμηλή πολυπλοκότητα και γενικά την αποδοτική χρήση του προγράμματος είναι οι πίνακες, μέσα στην τάξη LinkedList που προαναφέραμε, οι οποίοι περιέχουν τις διευθύνσεις που δείχνουν τον πρώτο κόμβο κάθε νέας ημέρας. Μέσα στον εσωτερικό βρόχο όταν συναντάμε την πρώτη τιμή μίας ημέρας αποθηκεύουμε τη διεύθυνση του κόμβου του εκάστοτε χρήστη στη θέση του πίνακα που

αντιστοιχεί στην ημέρα αυτή (π.χ. healthy1ptr[2] θα περιέχει τον πρώτο κόμβο της ημέρας 2 του χρήστη healthy1). Η χρησιμότητα των πινάκων αυτών βρίσκεται στις συναρτήσεις getters (getX, getY, getTime). Στις συναρτήσεις αυτές παίρνουμε ως όρισμα την τιμή που θέλουμε να αναζητήσουμε και την επιστρέφουμε στο χρήστη. Αρχικά δεν είχαμε υλοποιήσει τους πίνακες αυτούς και έτσι οι getters έψαχναν και μετρούσαν από τον πρώτο κόμβο της λίστας μέχρι την τιμή που θέλαμε να αναζητήσουμε. Αυτό είχε ως αποτέλεσμα ότι όσο αυξάνονταν οι μέρες, τόσες περισσότερες τιμές έπρεπε να μετρήσει μία συνάρτηση getter, δηλαδή για κάθε μέρα έκανε 2880 επαναλήψεις και αργούσε όλο και περισσότερο το πρόγραμμα να ανταποκριθεί. Όμως με τη χρήση αυτών των πινάκων ανάλογα σε ποια μέρα βρισκόμαστε, οι getters πηγαίνουν και μετρούν από τον πρώτο κόμβο εκείνης της ημέρας με αποτέλεσμα ο μέγιστος αριθμός επαναλήψεων να είναι 2880 (χειρότερη περίπτωση). Έτσι στους getters και συνάμα στις υπόλοιπες συναρτήσεις μειώθηκε κατά σημαντικό βαθμό ο χρόνος υλοποίησης του προγράμματος. Τα παραπάνω αποτέλεσαν θεμέλια για την πραγματοποιήση των τεσσάρων βασικών λειτουργιών του προγράμματος. Η πρώτη λειτουργία είναι η συνάρτηση POSSIBLE_COVID_19_INFECTION. Αυτή ελέγχει αν ο χρήστης βρέθηκε εντός ακτίνας R με κάποιον από τους δύο μολυσμένους χρήστες για τουλάχιστον T1 δευτερόλεπτα. Επίσης ελέγχει αν ο χρήστης βρέθηκε μετά από τουλάχιστον T2 δευτερόλεπτα σε ένα σημείο εντός της ακτίνας R που είχε βρεθεί και ένας μολυσμένος χρήστης. Ο έλεγχος αρχικά πραγματοποιείται για τα πρώτα 24*3600-T2 δευτερόλεπτα μέσω 2 βρόχων. Ο εξωτερικός εξετάζει όλες τις συντεταγμένες και με τη βοήθεια του εσωτερικού βρόχου ελέγχει αν θα ισχύσει η συνθήκη μετά από T2 δευτερόλεπτα. Για τα τελευταία T2 δευτερόλεπτα της ημέρας εξετάζει αν ισχύει η συνθήκη μέσω ενός βρόχου (ώστε να προσπελάσει τα υπόλοιπα στοιχεία) και μίας εντολής if. Ακόμη εξετάζει αν ισχύει η συνθήκη στο τέλος της ημέρας για να ενημερώσει το χρήστη ότι τελείωσε η ημέρα. Η συνάρτηση αυτή επιστρέφει true αν ο υγιής χρήστης βρέθηκε εντός ακτίνας R από τον μολυσμένο χρήστη για διάστημα τουλάχιστον T1 δευτερολέπτων της ώρας και το πολύ T2 δευτερόλεπτα αργότερα από τη στιγμή που πέρασε ο μολυσμένος, διαφορετικά επιστρέφει false. Η δεύτερη λειτουργία είναι η συνάρτηση FIND_CROWDED_PLACES η οποία επιστρέφει έναν ακέραιο αριθμό που συμβολίζει το πλήθος των χρηστών που βρέθηκαν εντός μίας τετραγωνικής περιοχής μία συγκεκριμένη ημέρα, εντός ενός χρονικού διαστήματος και παρέμειναν στην περιοχή τουλάχιστον για κάποια ώρα. Αυτή δέχεται ως ορίσματα την ημέρα που εξετάζουμε, την αρχή του χρονικού διαστήματος, το τέλος του, έναν πίνακα 4 στοιχείων, την ελάχιστη ώρα που πρέπει να μείνουν οι χρήστες ώστε να ικανοποιήσουν τη συνθήκη και τους 4 χρήστες της εφαρμογής μας. Η συνάρτηση αυτή εκτελείται κάθε μέρα και ο διαχειριστής της εφαρμογής πληκτρολογεί την αρχή και το τέλος του χρονικού διαστήματος, τις συντεταγμένες των 2 σημείων (αυτά τα 2 σημεία σχηματίζουν την τετραγωνική περιοχή, ζητώντας με κατάλληλο μήνυμα πρώτα τις συντεταγμένες του πάνω αριστερά στοιχείου A και ύστερα του κάτω δεξιά στοιχείου B) οι οποίες εκχωρούνται στον πίνακα. Επισημαίνεται ότι γίνονται οι κατάλληλοι έλεγχοι εγκυρότητας για τα δεδομένα που πληκτρολογεί ο χρήστης μέσω των συναρτήσεων Coordinates, Start, End, Tmin για τις συντεταγμένες, την αρχή του χρονικού διαστήματος, το τέλος του και την ελάχιστη χρονική διάρκεια αντίστοιχα. Η συνάρτηση FIND_CROWDED_PLACES αρχικά βάζει στα x1, y1, x2, y2 τις αντίστοιχες συντεταγμένες, αρχικοποιεί το μετρήτη count με 0 και ελέγχει μέσω ενός βρόχου από την αρχή του χρονικού διαστήματος μέχρι το τέλος του αν ο πρώτος χρήστης έμεινε στην τετραγωνική περιοχή. Για να βρίσκεται ο χρήστης μέσα στην τετραγωνική περιοχή πρέπει η τετμημένη του

να είναι ανάμεσα από τα x1, x2 καθώς και η τεταγμένη του ανάμεσα από τα y1, y2. Αν ισχύσει η συνθήκη αυξάνει τον μετρητή count κατά 1 και βγαίνει από το βρόχο (με την εντολή break). Η διαδικασία αυτή γίνεται και τους επόμενους 3 χρήστες και τελικά η συνάρτηση επιστρέφει το μετρήτη count, δηλαδή το πλήθος των χρηστών. Η τρίτη λειτουργία είναι η συνάρτηση REPAIR η οποία ουσιαστικά εισάγει κόμβους σε σημεία της λίστας που λείπουν διότι χάθηκε το σήμα GPS. Τη προσομοίωση της έλλειψης του σήματος GPS την πραγματοποιήσαμε μέσα στην συνάρτηση main με την μεταβλητή errorchance που έχει πιθανότητα περίπου μία φορά ανά δύο μέρες να μην εισάγει τον κόμβο του χρήστη για μία συγκεκριμένη χρονική στιγμή. Στο τέλος κάθε ημέρας εκτελείται η REPAIR για κάθε χρήστη και ελέγχει αν κάθε κόμβος της ημέρας εκείνης αναπαριστά τα δευτερόλεπτα που υπάρχουν σε αυτή την ημέρα ανά 30. Σε περίπτωση που ελέγξει κάποιο κόμβο και η ώρα time μέσα στον κόμβο δεν αντιστοιχεί στην σωστή ώρα εκείνης της ημέρας τότε εκτελείται η συνάρτηση Insert, η οποία εισάγει τον κόμβο που λείπει στην θέση αυτή με την σωστή πλέον ώρα και όσον αφορά τις συντεταγμένες παίρνουμε έναν μέσο όρο των συντεταγμένων του επόμενου και του προηγούμενου κόμβου. Η Insert πραγματοποιεί αυτήν την διαδικασία παίρνοντας ως όρισμα τον πρώτο κόμβο της ημέρας αυτής, και μέσω ενός βρόχου ψάχνει τη θέση στην οποία λείπει ο κόμβος και σε εκείνη τη θέση δημιουργεί έναν καινούριο κόμβο βάζοντας μέσα τα δεδομένα του χρήστη. Τέλος για να θεωρηθεί ως μέλος της λίστας κάνουμε τον δείκτη next του προηγούμενου κόμβου που δείχνει στην επόμενη διεύθυνση να δείχνει στη διεύθυνση του νέου κόμβου που εισήγαμε και τον next του κόμβου αυτού να δείχνει στη διεύθυνση του επόμενου κόμβου. Έτσι, πάντα είμαστε σίγουροι ότι ακόμα και αν χάθηκε το σήμα για κάποιο χρονικό διάστημα, το πρόγραμμα προσθέτει έναν κόμβο στο σημείο εκείνο ώστε να έχουμε πλήρη επίγνωση για την τροχιά του χρήστη. Η τέταρτη λειτουργία είναι η συνάρτηση SUMMARIZE_TRAJECTORY η οποία πηγαίνει λίγες μέρες πίσω από την μέρα στην οποία βρισκόμαστε και συνοψίζει την τροχιά ενός χρήστη. Δηλαδή αν έμενε για πολύ ώρα σε συντεταγμένες οι οποίες ήταν πολύ κοντά η μία στην άλλη, τις διαγράφει και εμφανίζει πιο συνοπτικά την τροχιά του χρήστη στον διαχειριστή της εφαρμογής. Τη συνάρτηση αυτή την εκτελούμε αφού το πρόγραμμα έχει τρέξει για μία εβδομάδα (7 μέρες), διότι για να εκτελεστεί χρειάζεται να έχουν ήδη εισαχθεί κόμβοι στην λίστα σε προηγούμενες μέρες. Για αυτό το λόγο με το που περάσει μία βδομάδα το πρόγραμμα από εκείνη τη στιγμή κάθε καινούρια μέρα πηγαίνει 7 μέρες πίσω και εκτελεί την SUMMARIZE_TRAJECTORY. Για παράδειγμα, όταν φτάσει για πρώτη φορά στην 7η ημέρα πηγαίνει 7 ημέρες πίσω στην ημέρα 0 του προγράμματος και συνοψίζει την τροχιά του χρήστη. Η διαδικασία της σύνοψης γίνεται πάλι παίρνοντας ως όρισμα τη διεύθυνση του πρώτου κόμβου της ημέρας στην οποία βρισκόμαστε μείον τις μέρες που θέλουμε να πάμε πίσω (Days – DaysBefore), και θεωρεί ως κέντρο ενός κύκλου με ακτίνα R τις συντεταγμένες του πρώτου κόμβου και εξετάζει αν η απόσταση των συντεταγμένων των επόμενων κόμβων σε σχέση με το κέντρο είναι μικρότερη είτε ίση με την ακτίνα. Αν είναι, τότε διαγράφουμε τον κόμβο που εξετάσαμε τελευταίο και προχωράμε στον επόμενο μέσω της συνάρτησης Delete, η οποία με παρόμοιο τρόπο με την Insert εντοπίζει τον κόμβο τον οποίο θέλουμε να διαγράψουμε και πάει απλά στον δείκτη next του προηγούμενου κόμβου από αυτόν και τον κάνει να δείχνει στην διεύθυνση του επόμενου κόμβου από αυτόν που θέλουμε να διαγράψουμε, έτσι φαίνεται σαν να τον έχουμε βγάλει από την αλυσίδα. Στην άλλη περίπτωση, δεν τον διαγράφουμε αλλά αντιθέτως τον εμφανίζουμε στον διαχειριστή και θεωρούμε τις συντεταγμένες του ως το νέο κέντρο του κύκλου και συνεχίζουμε και εξετάζουμε τους επόμενους κόμβους από αυτόν με την

ίδια διαδικασία μέχρι να τελειώσει η μέρα που εξετάζουμε. Αυτό έχει ως αποτέλεσμα να μην επαναλαμβάνονται κόμβοι που θα έχουν παρόμοιες ή ίδιες συντεταγμένες με τους προηγούμενους και διευκολύνουν το διαχειριστή της εφαρμογής να εμπεδώσει καλύτερα την τροχιά του κάθε χρήστη. Συνοψίζοντας, προσπαθήσαμε να βελτιστοποιήσουμε όσο το δυνατόν περισσότερο την απόδοση του προγράμματος. Προσθέσαμε ελέγχους εγκυρότητας όπου ήταν αναγκαίο και αξιοποιήσαμε συγκεκριμένες βιβλιοθήκες για ορισμένες διαδικασίες (δημιουργία τυχαίων τιμών, διαχείριση exception, κλπ). Το έγγραφο αυτό αποτελεί μια περίληψη των λειτουργιών του προγράμματός μας και για μία αναλυτικότερη περιγραφή μπορείτε να μεταβείτε στο αρχείο του πηγαίου κώδικα, όπου υπάρχουν «πυκνά» σχόλια που τεκμηριώνουν ακόμα καλύτερα το πρόγραμμα.