ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Project Δομών Δεδομένων

Μέλη Ομάδας:

Βασιλόπουλος Βασίλειος: up1097445

Χριστοδούλου Νικόλαος: up1093514

Χριστοδουλόπουλος Ευστάθιος Παναγιώτης: up1093513

Contents

Introduction

Για ποιον λόγο Rust;

Υπάρχουν αρκετοί λόγοι για τους οποίους επιλέξαμε να χρησιμοποιήσετε τη Rust αντί της C για το project μας. Ακολουθούν μερικά από τα βασικά πλεονεκτήματα της Rust:

1. Ασφάλεια μνήμης (**Borrow Checker**): Το βασικότερο χαρακτηριστικό διαφοροποίησης της Rust είναι η εστίασή της στην ασφάλεια της μνήμης. Επιβάλλει αυστηρούς κανόνες κατά τη μεταγλώττιση για την αποφυγή κοινών σφαλμάτων που σχετίζονται με τη μνήμη, όπως οι null pointer dereferences, οι υπερχειλίσεις του buffer και τα data races. Αυτά βοηθούν στην εξάλειψη ενός ευρέος φάσματος σφαλμάτων και καθιστά τα προγράμματα Rust πιο αξιόπιστα και ασφαλή σε σύγκριση με τη C.

2. Σύγχρονα εργαλεία και οικοσύστημα: Η Rust διαθέτει ένα ταχέως αναπτυσσόμενο οικοσύστημα με μια ενεργή κοινότητα, η οποία παρέχει εξαιρετικά εργαλεία και βιβλιοθήκες. Ο διαχειριστής πακέτων Cargo απλοποιεί τη διαχείριση εξαρτήσεων και τη ρύθμιση έργων, ενώ ο μεταγλωττιστής Rust παρέχει χρήσιμα μηνύματα σφαλμάτων και προειδοποιήσεις. Το οικοσύστημα Rust προσφέρει επίσης βιβλιοθήκες για ένα ευρύ φάσμα τομέων, διευκολύνοντας την αξιοποίηση του υπάρχοντος κώδικα και την επιτάχυνση της ανάπτυξης.

3. Επιδόσεις: Η Rust έχει σχεδιαστεί για να παρέχει έλεγχο χαμηλού επιπέδου στους πόρους του συστήματος, ενώ παράλληλα διασφαλίζει την ασφάλεια της μνήμης. Προσφέρει επιδόσεις συγκρίσιμες με τη C, χάρη σε χαρακτηριστικά όπως οι αφαιρέσεις μηδενικού κόστους και η ελάχιστη επιβάρυνση χρόνου εκτέλεσης. Η Rust το επιτυγχάνει αυτό αποφεύγοντας τους περιττούς ελέγχους κατά το χρόνο εκτέλεσης και επιτρέποντας λεπτομερή έλεγχο της διάταξης και της βελτιστοποίησης της μνήμης.

Παρόλο που η C είναι μια ώριμη και ευρέως χρησιμοποιούμενη γλώσσα με μια τεράστια υπάρχουσα βάση κώδικα, η Rust παρέχει μια σύγχρονη και ασφαλέστερη εναλλακτική λύση, ιδιαίτερα για τον προγραμματισμό συστημάτων, τα ενσωματωμένα συστήματα, τις κρίσιμες για τις επιδόσεις εφαρμογές και τα έργα που απαιτούν ισχυρές εγγυήσεις ασφάλειας μνήμης.

PART I:

Sorting and Searching Algorithms

Άσκηση (1)

Αλγόριθμοι Counting Sort και merge Sort

Ο κώδικας μας ξεκινά με την εισαγωγή των απαραίτητων modules και τον ορισμό μιας δομής `**Data**` που αναπαριστά μια σειρά δεδομένων από το αρχείο **CSV**. Με την συνάρτηση `**counting_sort**` υλοποιείται ταξινόμηση γραμμικού χρόνου **O(n+k)** στον vector των data. Ύστερα, η συνάρτηση `**merge_sort**` **O(nLogn)** διαιρεί αναδρομικά τον πίνακα εισόδου σε μικρότερους υποπίνακες μέχρι να φτάσει σε μια βασική περίπτωση με μήκος πίνακα <= 1 και στη συνέχεια εκτελεί το βήμα συγχώνευσης χρησιμοποιώντας τη συνάρτηση `**merge**`. Η συνάρτηση `**merge**` εκτελεί το βήμα συγχώνευσης στην ταξινόμηση, συγχωνεύοντας δύο ταξινομημένους υποπίνακες σε έναν ενιαίο ταξινομημένο πίνακα. Η συνάρτηση `**read_data**` δέχεται ένα όνομα αρχείου ως είσοδο, ανοίγει το αρχείο CSV χρησιμοποιώντας το crate `**csv**` και διαβάζει τα δεδομένα γραμμή προς γραμμή, αναλύοντας τις τιμές στη δομή `**Data**`. Η συνάρτηση επιστρέφει έναν **vector** από περιπτώσεις `**Data**` που αναπαριστούν τα δεδομένα από το αρχείο CSV. Για την διευκόλυνση μας προσθέσαμε τις βοηθητικές συναρτήσεις `**print_data**`, `**save_to_file**`, `**surround_with_quotes_if_comma**` και `**user_input**`. Τέλος στην συνάρτηση `**main**` το πρόγραμμα ξεκινά διαβάζοντας δεδομένα από ένα αρχείο **CSV** με όνομα "**effects.csv**" στον `**data_vector**`. Στη συνέχεια ο χρήστης καλείται να επιλέξει μεταξύ των αλγορίθμων ταξινόμησης counting sort ή merge sort. Με βάση την επιλογή του χρήστη, εκτελείται ο αντίστοιχος αλγόριθμος ταξινόμησης και υπολογίζεται ο χρόνος εκτέλεσης με τη χρήση της μονάδας του module `**SystemTime**` και τα ταξινομημένα δεδομένα εκτυπώνονται στην κονσόλα και εμφανίζεται ο χρόνος εκτέλεσης.

Με την πειραματική εκτέλεση των αλγορίθμων Counting Sort και Merge Sort παρατηρήσαμε ότι ο πρώτος είναι περίπου 24 φορές πιο αργός στο σύστημα μας. Αυτό εξηγείται από την δέσμευσης περισσοτέρων θέσεων μνήμης που κάνει ο Counting Sort έναντι στον Merge Sort.

Άσκηση (2)

Αλγόριθμοι Heap Sort και Quick Sort

Ο κώδικας μας ξεκινά με την εισαγωγή των απαραίτητων modules και τον ορισμό μιας δομής `**Data**` που αναπαριστά μια σειρά δεδομένων από το αρχείο **CSV**. Με την συνάρτηση `**heap_sort**` υλοποιεί τον αλγόριθμο ταξινόμησης σωρού **O(nLogn)** των στοιχείων `**Data**` με βάση το πεδίο `**cumulative**`. Η βοηθητική συνάρτηση `**heapify**` χρησιμοποιείται στον αλγόριθμο ταξινόμησης σωρού για τη διατήρηση της ιδιότητας του σωρού. Η συνάρτηση `**quick_sort**` **O(nLogn)** υλοποιεί τον αλγόριθμο γρήγορης ταξινόμησης των στοιχείων `**Data**` με βάση το πεδίο `**cumulative**`. Η συνάρτηση `**partition**` είναι μια επίσης βοηθητική συνάρτηση που χρησιμοποιείται στον αλγόριθμο γρήγορης ταξινόμησης για την κατάτμηση των δεδομένων γύρω από έναν άξονα. Η συνάρτηση `**read_data**` δέχεται ένα όνομα αρχείου ως είσοδο, ανοίγει το αρχείο CSV χρησιμοποιώντας το crate `**csv**` και διαβάζει τα δεδομένα γραμμή προς γραμμή, αναλύοντας τις τιμές στη δομή `**Data**`. Η συνάρτηση επιστρέφει έναν **vector** από περιπτώσεις `**Data**` που αναπαριστούν τα δεδομένα από το αρχείο CSV. Για την διευκόλυνση μας προσθέσαμε τις βοηθητικές συναρτήσεις `**print_data**`, `**save_to_file**`, `**surround_with_quotes_if_comma**` και `**user_input**`. Τέλος στην συνάρτηση `**main**` είναι το σημείο εισόδου του προγράμματος. Καλείται η συνάρτηση `**read_data**` για να διαβάσει τα δεδομένα από το αρχείο **CSV**, ζητά από τον χρήστη να

επιλέξει μεταξύ της ταξινόμησης σωρού και της γρήγορης ταξινόμησης και ανάλογα με την επιλογή του χρήστη καλείται η αντίστοιχη συνάρτηση ταξινόμησης (`**heap_sort**` ή `**quick_sort**`). Με την ολοκλήρωση της ταξινόμησης εκτυπώνει τα ταξινομημένα δεδομένα και το χρόνο εκτέλεσης του αλγορίθμου.

Με την πειραματική εκτέλεση των αλγορίθμων Heap Sort και Quick Sort παρατηρήσαμε ότι ο πρώτος είναι περίπου 2-3 φορές πιο γρήγορος στο σύστημα μας. Επειδή όμως η απόδοσή τους είναι παραπλήσια και σε επίπεδο millisecond, η διαφορά τους θεωρείται αμελητέα.

Άσκηση (3)

Δυαδικής Αναζήτησης και Αναζήτησης με Παρεμβολή.

Ο κώδικας μας ξεκινά με την εισαγωγή των απαραίτητων modules και τον ορισμό μιας δομής `**Data**` που αναπαριστά μια σειρά δεδομένων από το αρχείο **CSV**. Η συνάρτηση `**convert_date_to_days**` λαμβάνει μια συμβολοσειρά ημερομηνίας με τη μορφή "**dd/mm/yyyy**" και τη μετατρέπει σε συνολικό αριθμό ημερών. Η συνάρτηση `**binary_search**` εκτελεί δυαδική αναζήτηση στο διάνυσμα `**data**` για την εύρεση του δείκτη μιας συγκεκριμένης ημερομηνίας (`**date_key**`) χρησιμοποιώντας τη συνάρτηση `**convert_date_to_days**` για τη σύγκριση ημερομηνιών. Η συνάρτηση `**interpolation_search**` εκτελεί μια αναζήτηση παρεμβολής στο διάνυσμα `**data**` για να βρει το δείκτη μιας συγκεκριμένης ημερομηνίας (`**date_key**`) χρησιμοποιώντας παρεμβολή για προσεγγιστική τοποθέτηση. Η συνάρτηση `**read_data**` δέχεται ένα όνομα αρχείου ως είσοδο, ανοίγει το αρχείο CSV χρησιμοποιώντας το crate `**csv**` και διαβάζει τα δεδομένα γραμμή προς γραμμή, αναλύοντας τις τιμές στη δομή `**Data**`. Για την διευκόλυνση μας προσθέσαμε τις βοηθητικές συναρτήσεις `**print_data_line**`, `**in_range**` και `**user_input**`. Η συνάρτηση επιστρέφει έναν **vector** από περιπτώσεις `**Data**` που αναπαριστούν τα δεδομένα από το αρχείο CSV. Στη συνάρτηση `**main**` υπολογίζει τον χρόνο που απαιτείται για την ανάγνωση των δεδομένων από το αρχείο CS και ζητά από τον χρήστη να εισάγει μια ημερομηνία. Εάν η εισαγόμενη ημερομηνία είναι εκτός του εύρους των ημερομηνιών στο σύνολο δεδομένων, εκτυπώνει "**Date out of range**" (Ημερομηνία εκτός εύρους). Αλλιώς μετατρέπει την ημερομηνία εισαγωγής σε ημέρες για ταχύτερη σύγκριση και ταυτόχρονα εκτελεί δυαδική αναζήτηση και αναζήτηση παρεμβολής στον vector `**data**` για να βρει τον δείκτη της ημερομηνίας εισόδου. Τέλος εκτυπώνει την αντίστοιχη γραμμή δεδομένων καθώς και τον χρόνο αναζήτησης του κάθε αλγορίθμου αναζήτησης.

Και οι δύο αλγόριθμοι έχουνε time complexity O(**logn**) με την διαφορά ότι η αναζήτηση με παρεμβολή έχει χειρότερη περίπτωση O(**n**). Το γεγονός ότι έχουμε ομοιόμορφα κατανεμημένα δεδομένα δίνει ένα πλεονέκτημα στον αλγόριθμο που βασίζεται στην λογική των πιθανοτήτων δηλαδή στην αναζήτηση με παρεμβολή.

Άσκηση (4)

Δυική Αναζήτηση Παρεμβολής (BIS)

Ο κώδικας μας ξεκινά με την εισαγωγή των απαραίτητων modules και τον ορισμό μιας δομής `**Data**` που αναπαριστά μια σειρά δεδομένων από το αρχείο **CSV**. Η συνάρτηση `**date_to_days**` λαμβάνει μια συμβολοσειρά ημερομηνίας με τη μορφή "**dd/mm/yyyy**" και τη μετατρέπει σε συνολικό αριθμό ημερών. Η συνάρτηση `**read_data**` δέχεται ένα όνομα αρχείου ως είσοδο, ανοίγει το αρχείο CSV χρησιμοποιώντας το crate `**csv**` και διαβάζει τα δεδομένα γραμμή προς γραμμή, αναλύοντας τις τιμές στη δομή `**Data**`. Για την διευκόλυνση μας προσθέσαμε τις βοηθητικές συναρτήσεις `**print_data_line**`, `**in_range**` και `**user_input**`. Η συνάρτηση επιστρέφει έναν **vector** από περιπτώσεις `**Data**` που αναπαριστούν τα

δεδομένα από το αρχείο CSV. Η συνάρτηση `**bis**` υλοποιεί έναν αλγόριθμο δυαδικής αναζήτησης για την αποτελεσματική εύρεση μιας εγγραφής με συγκεκριμένη ημερομηνία σε ένα ταξινομημένο σύνολο δεδομένων. Αρχικοποιεί τα όρια, εκτιμά έναν δείκτη με βάση την ημερομηνία-στόχο και εισέρχεται σε έναν βρόχο συγκρίνοντας τον στόχο με την ημερομηνία στον εκτιμώμενο δείκτη. Προσαρμόζει τα όρια και συνεχίζει την αναζήτηση στο άνω ή στο κάτω μισό του υπόλοιπου συνόλου δεδομένων μέχρι να βρεθεί η ημερομηνία-στόχος ή να εξαντληθεί ο χώρος αναζήτησης. Η συνάρτηση επιστρέφει `**true**` και τον δείκτη αν η ημερομηνία βρεθεί, ή `**false**` και τον δείκτη 0 αν δεν βρεθεί. Η συνάρτηση `**main**` είναι το σημείο εισόδου του προγράμματος. Καλεί τη συνάρτηση `**read_data**` για να διαβάσει τα δεδομένα από το αρχείο "**cs.csv**". Στη συνέχεια ζητά από τον χρήστη να δώσει μια ημερομηνία χρησιμοποιώντας τη συνάρτηση `**user_input**`. Εάν η ημερομηνία εισόδου είναι εκτός εύρους σύμφωνα με τη συνάρτηση `**in_range**`, εκτυπώνει "**Date out of range**" (Ημερομηνία εκτός εύρους) και εξέρχεται από το πρόγραμμα. Διαφορετικά, εκκινεί το χρονόμετρο χρησιμοποιώντας τη συνάρτηση `**SystemTime::now()**` και καλεί τη συνάρτηση `**bis**` για να αναζητήσει την ημερομηνία εισόδου στο σύνολο δεδομένων. Εάν η ημερομηνία βρεθεί, εκτυπώνει το δείκτη και την αντίστοιχη εγγραφή `**Data**`. Εκτυπώνει επίσης τον χρόνο που χρειάστηκε για την αναζήτηση. Εάν η ημερομηνία δεν βρεθεί, εκτυπώνει "**Date not found**" (Ημερομηνία δεν βρέθηκε).

Ως προς τους χρόνους χειρότερης περίπτωσης το time complexity του αλγορίθμου της δυαδικής παρεμβολής είναι O(n).

PART II:

BSTs && HASHING

Άσκηση (A)

Δένδρο BST

Ο κώδικας μας ξεκινά με την εισαγωγή των απαραίτητων modules και τον ορισμό δύο δομών δεδομένων `**Data**` και `**Node**`. Η δομή `**Data**` αναπαριστά μια ενιαία εγγραφή εμπορικών δεδομένων με τα διάφορα πεδία της. Η δομή `**Node**` αναπαριστά έναν κόμβο στο δέντρο **AVL**. Περιέχει μια αναφορά σε μια εγγραφή δεδομένων (`**Data**`), καθώς και αριστερούς και δεξιούς κόμβους-παιδιά και μια τιμή ύψους που χρησιμοποιείται για την εξισορρόπηση του δέντρου. Στη συνέχεια, ορίζεται η δομή `**AvlTree**`, η οποία χρησιμεύει ως η κύρια δομή δεδομένων του δέντρου **AVL**. Διαθέτει μεθόδους για την αρχικοποίηση ενός νέου δέντρου, την εισαγωγή δεδομένων στο δέντρο, την εκτέλεση μιας διάσχισης του δέντρου κατά σειρά, την αναζήτηση δεδομένων με βάση την ημερομηνία, τη διαγραφή δεδομένων με βάση την ημερομηνία και την επεξεργασία δεδομένων με βάση την ημερομηνία. Ο κώδικας περιλαμβάνει επίσης διάφορες βοηθητικές συναρτήσεις όπως η `**date_to_days**` η οποία μετατρέπει μια συμβολοσειρά ημερομηνίας στον αριθμό των ημερών από το έτος 0, η `**height**` η οποία επιστρέφει το ύψος ενός κόμβου και διάφορες συναρτήσεις για την εξισορρόπηση του δέντρου **AVL**. Οι συναρτήσεις `**insert**`, `**inorder**`, `**search_node**`, `**delete_node**` και `**edit_node**` είναι υλοποιήσεις των αντίστοιχων μεθόδων της δομής `**AvlTree**`. Αυτές οι συναρτήσεις εκτελούν τις πραγματικές λειτουργίες στο δέντρο. Η συνάρτηση `**read_data**` χρησιμοποιείται για την ανάγνωση δεδομένων συναλλαγών από ένα αρχείο **CSV** και την κατασκευή ενός δέντρου **AVL** που περιέχει τα δεδομένα. Η συνάρτηση `**print_data**` είναι μια βοηθητική συνάρτηση για την εκτύπωση των περιεχομένων μιας εγγραφής εμπορικών δεδομένων και συνάρτηση `**user_input**` χρησιμοποιείται για να διαβάζει την είσοδο του χρήστη από την κονσόλα. Τέλος, στη συνάρτηση `**main**`, ο κώδικας διαβάζει δεδομένα από ένα αρχείο **CSV**, κατασκευάζει ένα δέντρο **AVL** και

εισέρχεται σε έναν βρόχο όπου ο χρήστης μπορεί να εκτελέσει λειτουργίες στο δέντρο, όπως διάσχιση, αναζήτηση, επεξεργασία και διαγραφή.

Άσκηση (Β)

Δένδρο BST με max/min value

Ο κώδικας μας ξεκινά με την εισαγωγή των απαραίτητων ενοτήτων `**File**`, `**Write**`, `**exit**`, `**SystemTime**` και τον ορισμό δύο δομών δεδομένων `**Data**` και `**Node**`. Το `**Data**` αναπαριστά μια απλή εγγραφή δεδομένων, ενώ το `**Node**` αναπαριστά έναν κόμβο στο δέντρο **AVL**.Ύστερα Ορίζεται η δομή `**AvlTree**`, η οποία χρησιμεύει ως δομή δεδομένων του δέντρου **AVL** και περιέχει έναν κόμβο ρίζας. Το μπλοκ `**impl**` ορίζει μεθόδους για τη δομή `**AvlTree**`, η μέθοδος `**new**` δημιουργεί μια νέα περίπτωση του δέντρου **AVL** και η μέθοδος `**insert**` εισάγει μια νέα εγγραφή δεδομένων στο δέντρο. Αρκετές βοηθητικές συναρτήσεις ορίζονται εκτός του μπλοκ `**impl**`. Αυτές οι συναρτήσεις χρησιμοποιούνται για λειτουργίες του δέντρου AVL, όπως ο υπολογισμός του ύψους των κόμβων, των συντελεστών ισορροπίας και η εκτέλεση περιστροφών. Η συνάρτηση `**insert**` εισάγει αναδρομικά έναν νέο κόμβο στο δέντρο **AVL** με βάση την τιμή της εγγραφής δεδομένων. Δύο συναρτήσεις, `**node_with_min_value**` και `**node_with_max_value**`, διασχίζουν το δέντρο **AVL** για να βρουν τους κόμβους με την ελάχιστη και τη μέγιστη τιμή, αντίστοιχα. Η συνάρτηση `**nodes_with_same_value**` διασχίζει αναδρομικά το δέντρο **AVL** για να βρει κόμβους με την ίδια τιμή με την παρεχόμενη τιμή και συλλέγει αυτούς τους κόμβους σε έναν vector. Η συνάρτηση `**read_data**` διαβάζει δεδομένα από ένα αρχείο **CSV** και κατασκευάζει ένα δέντρο **AVL** χρησιμοποιώντας τη μέθοδο `**insert**`. Η συνάρτηση `**user_input**` διαβάζει μια γραμμή εισόδου από τον χρήστη και την επιστρέφει ως συμβολοσειρά. Η συνάρτηση `**print_data**` εκτυπώνει τα πεδία μιας δομής `**Data**`. Όσον αφορά την `**main**`, καλείται η συνάρτηση `**read_data**` για να διαβάσει δεδομένα από το αρχείο **CSV** και να κατασκευάσει το δέντρο **AVL**. Έπειτα ξεκινάει ένας βρόχος για την αλληλεπίδραση με τον χρήστη. Ο χρήστης μπορεί να επιλέγει από το **TUI** την εύρεση δεδομένων είτε με τη μέγιστη τιμή είτε την ελάχιστη τιμή ή διαφορετικά την έξοδο από το πρόγραμμα. Με βάση την επιλογή του χρήστη, καλούνται οι αντίστοιχες συναρτήσεις για την εύρεση και την εκτύπωση των δεδομένων. Ο βρόχος συνεχίζεται μέχρι ο χρήστης να επιλέξει την έξοδο.

Άσκηση (Γ)

HASHING με αλυσίδες

Ο κώδικας μας ξεκινά με την εισαγωγή των απαραίτητων ενοτήτων `**File**`, `**Write**`, `**exit**`, `**SystemTime**`. Η σταθερά `**MOD**`, η οποία αντιπροσωπεύει το μέγεθος του πίνακα κατακερματισμού ορίζεται ως 11, η οποία αντιπροσωπεύει το μέγεθος του πίνακα κατακερματισμού. Ύστερα ορίζονται δύο δομές `**Data**` και `**Node**`. Η δομή `**Data**` αναπαριστά τα δεδομένα που είναι αποθηκευμένα σε κάθε κόμβο του συνδεδεμένου καταλόγου. Η δομή `**Node**` αναπαριστά έναν κόμβο της συνδεδεμένης λίστας και περιέχει τα δεδομένα και μια αναφορά στον επόμενο κόμβο. Η δομή `**LinkedList**` ορίζεται για την αναπαράσταση μιας συνδεδεμένης λίστας. Περιέχει μια αναφορά στον πρώτο κόμβο `**first**` και μια αναφορά στον τελευταίο κόμβο `**last**`. Οι υλοποιήσεις για τις δομές ορίζονται χρησιμοποιώντας τα μπλοκ `**impl**`. Αυτές οι υλοποιήσεις περιλαμβάνουν μεθόδους για τη δημιουργία νέων περιπτώσεων των structs και την εκτέλεση πράξεων σε αυτές. Η συνάρτηση `**init**` αρχικοποιεί τον πίνακα κατακερματισμού, ο οποίος είναι ένα διάνυσμα συνδεδεμένων λιστών, με κενές συνδεδεμένες λίστες. Επιστρέφει τον αρχικοποιημένο πίνακα κατακερματισμού. Η συνάρτηση `**hash**` υπολογίζει την τιμή κατακερματισμού για μια δεδομένη συμβολοσειρά ημερομηνίας. Επαναλαμβάνει κάθε χαρακτήρα στη συμβολοσειρά και προσθέτει την τιμή

Unicode του στο άθροισμα. Στη συνέχεια, το άθροισμα διαιρείται με τη σταθερά `**MOD**` για να προκύψει η τιμή κατακερματισμού. Η συνάρτηση `**insert**` δέχεται ως παραμέτρους έναν πίνακα κατακερματισμού και δεδομένα. Υπολογίζει την τιμή κατακερματισμού για την ημερομηνία των δεδομένων και εισάγει τα δεδομένα στην αντίστοιχη συνδεδεμένη λίστα του πίνακα κατακερματισμού. Η συνάρτηση `**search**` δέχεται ως παραμέτρους έναν πίνακα κατακερματισμού και μια συμβολοσειρά ημερομηνίας. Αναζητά τον κόμβο με τη δεδομένη ημερομηνία στην αντίστοιχη συνδεδεμένη λίστα του πίνακα κατακερματισμού και επιστρέφει τον κόμβο αν βρεθεί. Η συνάρτηση `**edit**` δέχεται ως παραμέτρους έναν πίνακα κατακερματισμού, μια συμβολοσειρά ημερομηνίας και νέα δεδομένα. Αναζητά τον κόμβο με τη δεδομένη ημερομηνία στην αντίστοιχη συνδεδεμένη λίστα του πίνακα κατακερματισμού και ενημερώνει τα δεδομένα του κόμβου με τα νέα δεδομένα. Η συνάρτηση `**delete_and_attach_next**` λαμβάνει έναν προαιρετικό κόμβο και αποσυνδέει τον επόμενο κόμβο του (αν υπάρχει) και τον επιστρέφει. Η συνάρτηση `**delete**` δέχεται ως παραμέτρους έναν πίνακα κατακερματισμού και μια συμβολοσειρά ημερομηνίας. Αναζητά τον κόμβο με τη δεδομένη ημερομηνία στην αντίστοιχη συνδεδεμένη λίστα του πίνακα κατακερματισμού. Αν βρεθεί, διαγράφει τον κόμβο και συνδέει τον επόμενο κόμβο (αν υπάρχει) με τον προηγούμενο κόμβο. Η συνάρτηση `**read_data**` διαβάζει δεδομένα από ένα αρχείο **CSV** (που καθορίζεται από την παράμετρο `**filename**`) και αρχικοποιεί τον πίνακα κατακερματισμού με τα δεδομένα που έχουν διαβαστεί. Χρησιμοποιεί το crate `**csv**` για να αναλύσει το αρχείο **CSV** και δημιουργεί περιπτώσεις `**Data**` από κάθε εγγραφή. Η συνάρτηση `**insert**` καλείται για να εισάγει κάθε στοιχείο δεδομένων στον πίνακα κατακερματισμού. Η συνάρτηση `**user_input**` διαβάζει μια γραμμή εισόδου από τον χρήστη και την επιστρέφει ως συμβολοσειρά. Η συνάρτηση `**print_data**` λαμβάνει μια αναφορά σε μια δομή `**Data**` και εκτυπώνει τα πεδία της. Η συνάρτηση `**main**` διαβάζει δεδομένα από το αρχείο **CSV** χρησιμοποιώντας τη συνάρτηση `**read_data**` και αρχικοποιεί τον πίνακα κατακερματισμού. Στη συνέχεια εισέρχεται σε έναν βρόχο που παρουσιάζει ένα μενού στον χρήστη και εκτελεί λειτουργίες με βάση τις επιλογές του. Οι επιλογές του μενού περιλαμβάνουν αναζήτηση, επεξεργασία, διαγραφή και έξοδο. Η επιλογή του χρήστη διαβάζεται με τη χρήση της `**user_input**` και καλούνται οι αντίστοιχες συναρτήσεις για την εκτέλεση των πράξεων στον πίνακα κατακερματισμού.