**Τεχνική Έκθεση – Υλοποίηση Δομών Δεδομένων σε C++**

Η παρούσα εργασία έχει ως στόχο την υλοποίηση μιας βιβλιοθήκης δομών δεδομένων σε C++ για τη διαχείριση διαφόρων συλλογών στοιχείων. Οι δομές αυτές χρησιμοποιούνται ευρέως σε αλγοριθμικές και πρακτικές εφαρμογές, βελτιώνοντας την αποδοτικότητα της επεξεργασίας δεδομένων.

**ΣΩΡΟΙ**

Τα δέντρα σωροί είναι μια υποκατηγορία των δυαδικών δέντρων και χωρίζονται σε 2 ειδή, στους σωρούς ελάχιστων και μέγιστων. Σε έναν σωρό ελάχιστων κάθε στοιχείο ενός κόμβου είναι μικρότερο από τα στοιχεία των παιδιών του. Ενώ για να χαρακτηριστεί ένα δυαδικό δέντρο ως σωρός μέγιστων πρέπει κάθε στοιχείο ενός κόμβου είναι μεγαλύτερο από τα στοιχεία των παιδιών του. Στην μνήμη αναπαριστονται από ένα δυναμικό μονοδιαστατο πινακα.

Εικόνα που περιέχει κύκλος, στιγμιότυπο οθόνης

Το περιεχόμενο που δημιουργείται από AI ενδέχεται να είναι εσφαλμένο.

ΑΝΑΠΑΡΑΣΤΑΣΗ ΣΤΗΝ ΜΝΗΜΗ

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **1** | **3** | **2** | **4** | **6** | **5** |  |
| **6** | **5** | **3** | **4** | **2** | **1** |  |

**MINHEAP:**

**MAXHEAP:**

Γονιός: (i-1) DIV 2

Αριστερό παιδί: 2\*ι + 1

Δεξί παιδί : 2\*i + 2

|  |  |  |
| --- | --- | --- |
| ΑΛΓΟΡΥΘΜΟΣ | ΠΕΡΙΓΡΑΦΗ | ΠΟΛΥΠΛΟΚΟΤΗΤΑ |
| BuildHeapFromFile | Η συνάρτηση BuildHeapFromFile λαμβάνει το όνομα ενός αρχείου, ανοίγει το αρχείο και διαβάζει ακέραιους αριθμούς. Κάθε αριθμός εισάγεται στον πίνακα και στη συνέχεια καλείται η heapifyDown από το μέσο του πίνακα έως την αρχή για να διασφαλιστεί η ιδιότητα του ελάχιστου σωρού (bottom-up προσέγγιση). | O(N) |
| insert | Η συνάρτηση insert εισάγει ένα νέο στοιχείο στο τέλος του πίνακα και με τη μέθοδο heapifyUp εξασφαλίζει ότι η ιδιότητα του ελάχιστου σωρού διατηρείται μετά την εισαγωγή. | Ο(logN) |
| getMin | Η getMin επιστρέφει το στοιχείο που βρίσκεται στην κορυφή του σωρού, δηλαδή το μικρότερο στοιχείο. | O(1) |
| deleteMin | Η deleteMin αφαιρεί το στοιχείο με τη μικρότερη τιμή, το ανταλλάσσει με το τελευταίο στοιχείο του πίνακα και στη συνέχεια καλεί τη heapifyDown για να αναδιατάξει τον σωρό ώστε το καινούργιο ελάχιστο στοιχείο να μετακινηθεί στην κορυφή. | O(logN) |
| getSize | Η getSize επιστρέφει το πλήθος των στοιχείων του πίνακα, δηλαδή πόσα στοιχεία περιέχει αυτή τη στιγμή ο σωρός. | O(1) |
| heapifyUp | Η heapifyUp εξετάζει έναν κόμβο και το συγκρίνει με τον γονέα του. Αν το στοιχείο είναι μικρότερο / μεγαλύτερο από τον γονέα, γίνεται η απαιτούμενη ανταλλαγή και προχωράμε προς τα πάνω μέχρι να αποκατασταθεί η ιδιότητα του σωρού. | O(logN) |
| heapifyDown | Η heapifyDown εξετάζει έναν κόμβο και τον συγκρίνει με τα παιδιά του. Αν κάποιο από τα παιδιά του είναι μικρότερο/μεγαλύτερο από τον κόμβο, γίνεται η απαιτούμενη ανταλλαγή και προχωράμε προς τα κάτω μέχρι να αποκατασταθεί η ιδιότητα του σωρού. | O(logN) |

Απόδειξη πολυπλοκότητας της τεχνικής bottom up

( Οι υπόλοιπες είναι εύκολες και παραλείπονται)

Αν ορίζουμε το “ύψος” ενός κόμβου ως την απόστασή του (σε ακμές) από τα φύλλα, τότε καλώντας την heapifyDown σε κόμβο ύψους h, στο χειρότερο σενάριο το στοιχείο «κατρακυλά» ως τα φύλλα, δηλαδή κάνει το πολύ h συγκρίσεις/ανταλλαγές



Σε ένα σωρό με n κόμβους, ο αριθμός των κόμβων ύψους h είναι το πολύ

Εικόνα που περιέχει κείμενο, γραμματοσειρά, σχεδίαση, τυπογραφία

Το περιεχόμενο που δημιουργείται από AI ενδέχεται να είναι εσφαλμένο.

Γιατί κάθε επίπεδό έχει τους μίσους κόμβους από το επόμενο και

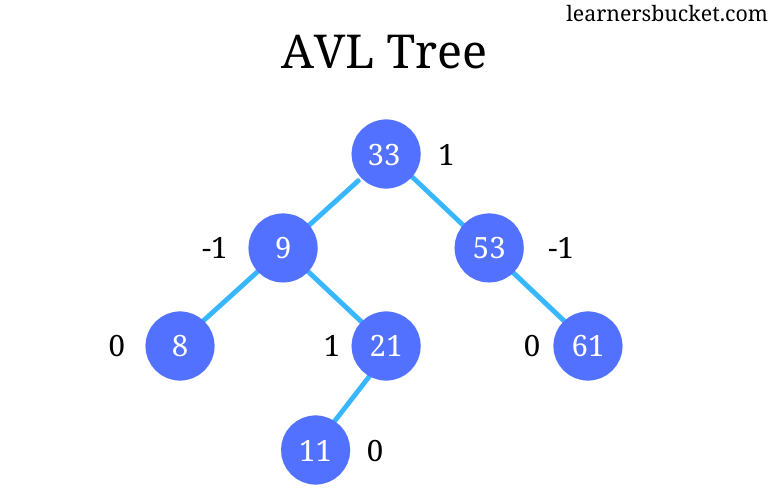
n<=2^(h+1)-1<= 2^(h+1)

Εικόνα που περιέχει κείμενο, γραμματοσειρά, γραφικός χαρακτήρας, λευκό

Το περιεχόμενο που δημιουργείται από AI ενδέχεται να είναι εσφαλμένο.

Αποδεικνύεται ότι η σειρά είναι συγκλίνει στο 1 πολύ ευκολά (αν παραγωγήσουμε την σειρά Σ x^h)

**AVL** **TREE**

Τα δέντρα AVL είναι μια υποκατηγορία των δυαδικών δέντρων αναζήτησης τα οποία είναι ισοζυγισμένα. Αυτό σημαίνει ότι η διαφορά των υψών του αριστερού και του δεξιού υποδένδρου δεν πρέπει να ξεπερνά το 1 και αυτό να ισχύει αναδρομικά για όλους τους κόμβους του δένδρου. Απώτερος σκοπός είναι η διαδικασία της αναζήτησης να δικπαιρεωνεται με πολυπλοκότητα logn.

Απόδειξη πολυπλοκότητας του αλγορίθμου αναζήτησης:

Στο χειρότερο σενάριο θα γίνουν τόσες συγκρίσεις όσο και το συνολικό ύψος του δέντρου h. Αφού το δέντρο είναι καλά ζυγισμένο έχει το πολύ περίπου 2^(h+1)-1 κόμβους. Δηλαδή

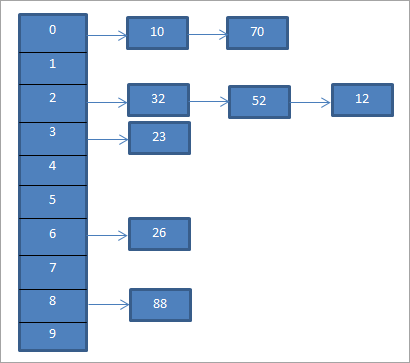
N= 2^(h+1)-1 ⬄

⬄logN = O(h)

Αρά η πολυπλοκότητα είναι logN.

|  |  |  |
| --- | --- | --- |
| ΑΛΓΟΡΥΘΜΟΣ | ΠΕΡΙΓΡΑΦΗ | ΠΟΛΥΠΛΟΚΟΤΗΤΑ |
| buildFromFile | Η συνάρτηση buildFromFile διαβάζει από ένα αρχείο ακέραιους αριθμούς και τους εισάγει διαδοχικά στο AVL δέντρο. Χρησιμοποιεί τη συνάρτηση insertKey. | Ο(NlogN) |
| insertNode | Η συνάρτηση insertNode εισάγει έναν νέο κόμβο με τιμή key στο AVL δέντρο. Eνώ παραλληλα χρησιμοποιεί μια στοίβα (Vector<AVLNode\*>) για να εντοπίζει γρήγορα τους προγόνους, και να προβαίνει στις περιστροφές που χρειάζονται, ώστε να διατηρηθεί η ιδιότητα του AVL. | O(logN) |
| deleteNode | Η συνάρτηση deleteNode διαγράφει από το δέντρο τον κόμβο με τιμή key, αν υπάρχει. Αν διαπιστωθεί κάποια παράβαση της ιδιότητας AVL σε κάποιον από τους προγονούς του διαγραφομένου κόμβου προβαίνει στις κατάλληλες περιστροφές (LL,LR,RL,RR). | Ο(logN) |
| search | Η συνάρτηση search αναζητά ένα κλειδί στο ΑVL δέντρο. Επιστρέφει true αν το κλειδί βρεθεί, αλλιως false. Εφαρμόζεται η μέθοδος της δυαδικής αναζήτησης. | O(logN) |
| findMinNode | Η συνάρτηση findMinNode αναζητά και επιστρέφει τον κόμβο που περιέχει το ελάχιστο κλειδί σε ένα υποδέντρο. Αν το υποδέντρο είναι κενό, επιστρέφει nullptr. | O(logN) |
| getSize | Η getSize επιστρέφει το τρέχον πλήθος των κόμβων που υπάρχουν στο AVL δέντρο. | O(1) |
| rotateLeft | Η συνάρτηση rotateLeft εκτελεί μια απλή περιστροφή προς τα αριστερά σε έναν κόμβο y, επανατοποθετώντας κατάλληλα τους δεσμούς για να επανέλθει η ισορροπία του δέντρου. | O(1) |
| rotateRight | Η συνάρτηση rotateRight εκτελεί μια απλή περιστροφή προς τα δεξιά σε έναν κόμβο x, επανατοποθετώντας κατάλληλα τους δεσμούς για να επανέλθει η ισορροπία του δέντρου. | O(1) |

**ΠΙΝΑΚΑΣ ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΥ**

Ένας πίνακας κατακερματισμού ( hashtable ) είναι μία δομή δεδομένων που αποθηκεύει στοιχεία ή ζευγάρια στοιχείων, επιτρέποντας γρήγορη εισαγωγή, αναζήτηση και διαγραφή στοιχείων με μέσο χρόνο O(1), υπό ορισμένες προϋποθέσεις. Ουσιαστικά αν υλοποιήσουμε την δομή με την τεχνική της αλυσίδας, φτιάχνουμε έναν μονοδιάστατο πίνακα από απλά συνδεδεμένες λίστες και στην συνέχεια για κάθε εισαγωγή στοιχείου με την βοήθεια μιας hash συνάρτησης επιλέγουμε την λίστα στην οποία θα προσθέσουμε το στοιχείο.

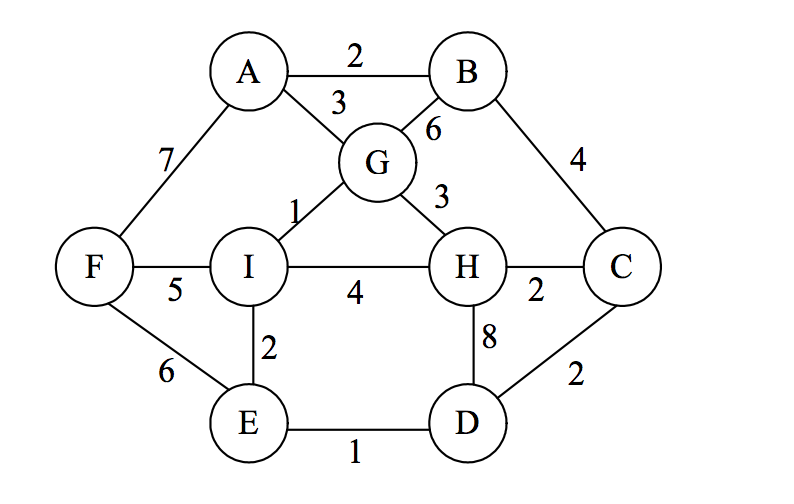
Hash function -> Προσπαθεί να διακρίνει τα στοιχεία σε κλάσεις. Όσο πιο ομοιόμορφα γίνει η κατανομή των στοιχείων τόσο καλύτερη είναι η συνάρτηση κατακερματισμού.

Capacity -> Το μέγεθος του μονοδιάστατου πίνακα.

|  |  |  |
| --- | --- | --- |
| ΑΛΓΟΡΥΘΜΟΣ | ΠΕΡΙΓΡΑΦΗ | ΠΟΛΥΠΛΟΚΟΤΗΤΑ |
| BuildHeapFromFile | Η buildFromFile καθαρίζει το table και στη συνέχεια, διαβάζει ακεραίους από ένα αρχείο και καλεί τη συνάρτηση insertKey για καθέναν από αυτούς. | O(N) |
| hashFunction | Η συνάρτηση hashFunction υπολογίζει το index για ένα κλειδί (key) εκτελώντας πράξη modulus (% capacity). | Ο(1) |
| getSize | Η getSize επιστρέφει το τρέχον μέγεθος της δομής κατακερματισμού (size), δηλαδή τον αριθμό των στοιχείων που περιέχονται στον πίνακα κατακερματισμού. | O(1) |
| searchKey | Η searchKey ελέγχει αν ένα κλειδί υπάρχει στον πίνακα κατακερματισμού. Υπολογίζει το index μέσω της hashFunction, και κατόπιν καλεί τη μέθοδο contains στην αντίστοιχη λίστα (LinkedList<int>) έτσι ώστε να διαπιστώσει αν περιέχει το key. | ~O(1)  Υπό προϋποθέσεις |
| insertKey | Η insertKey εισάγει ένα νέο κλειδί στον πίνακα, an δεν υπάρχει ήδη. Υπολογίζει το hash και εισάγει το key στην κατάλληλη λίστα. Αυξάνει το size | ~O(1) |
| deleteKey | Η deleteKey διαγράφει ένα κλειδί από τον πίνακα κατακερματισμού, υπολογίζοντας το index του μέσω της hashFunction και καλώντας τη remove της αντίστοιχης λίστας (LinkedList<int>) για να αφαιρέσει το key. | ~O(1) |
| getAllVertices  *Στην ειδική έκδοση της δομής για τους γράφους HASHTABLEPAIRS* | Η getAllVertices επιστρέφει ένα Vector<int> με όλες τις κορυφές (vertexIds) που υπάρχουν στη δομή. | O(N + M)  M: το πλήθος των ακμών |

**ΓΡΑΦΟΙ**

Ένας γράφος αποτελείται από κόμβους (ή κορυφές) που συνδέονται μεταξύ τους μέσω ακμών. Όταν οι ακμές έχουν κατεύθυνση, δηλαδή ορίζονται από διατεταγμένα ζεύγη κόμβων, ονομάζεται κατευθυνόμενος γράφος. Αντίθετα, όταν οι ακμές δεν έχουν κατεύθυνση πρόκειται για μη κατευθυνόμενο γράφο. Επιπλέον, εάν σε κάθε ακμή του γράφου αντιστοιχεί ένα βάρος τότε ο γράφος χαρακτηρίζεται ως σταθμισμένος. Εδώ έχουμε υλοποιήσει γράφημα χωρίς κατευθύνσεις αλλά με βάρη στις ακμές.



**Στην μνήμη αναπαρίσταται από έναν πίνακα κατακερματισμού και λίστες γειτνίασής για τις αντίστοιχες ακμές. Αυτή η αναπαράσταση ευνοεί πολύ αλγορίθμους όπως ο Dijkstra ενώ είναι ιδανικό για scale.**

|  |  |  |
| --- | --- | --- |
| ΑΛΓΟΡΥΘΜΟΣ | ΠΕΡΙΓΡΑΦΗ | ΠΟΛΥΠΛΟΚΟΤΗΤΑ |
| buildFromFile | Η συνάρτηση Graph::buildFromFile διαβάζει ένα αρχείο κειμένου και κατασκευάζει το γράφο προσθέτοντας διαδοχικά τις ακμές που περιγράφονται σε αυτό. | Ο(N) |
| addVertex | Προσθέτει μια νέα κορυφή, αν δεν υπάρχει ήδη, στην εσωτερική δομή κατακερματισμού | O(1) |
| addEdge | Προσθέτει μία ακμή (source->destination) και άλλη μία αντίστροφη (destination->source). Αν η ακμή υπάρχει ήδη, απλώς δεν την προσθέτει ξανά. | Ο(Μ)  Λόγο της hasEdge |
| hasEdge | Ψάχνουμε μέσα στη λίστα ακμών (srcEdges) να βρούμε αν υπάρχει ακμή προς το destination. | O(Μ) |
| getEdgeWeight | Επιστρέφει το βάρος της ακμής (source->destination) ή -1, αν δεν υπάρχει τέτοια ακμή | ~O(Μ) |
| removeEdge | Αν υπάρχει ακμή ανάμεσα στο source και το destination, την αφαιρούμε και από τις δύο αντίστοιχες λίστες ακμών κι ενημερώνουμε το numEdges. | ~O(Μ) |
| getNeighbors | Επιστρέφει όλους τους γείτονες μίας κορυφής, δηλαδή όλες τις κορυφές που συνδέονται απευθείας με αυτήν. Αν η κορυφή δεν υπάρχει, επιστρέφουμε έναν κενό Vector<int>. | O(M)  M: το πλήθος των ακμών |
| getNumVertices | Επιστρέφει το πλήθος των κορυφών που υπάρχουν στον γράφο | O(1) |
| getNumEdges | Επιστρέφει τον αριθμό των ακμών (numEdges) που υπάρχουν στον γράφο | O(1) |
| computeShortestPath | Υλοποίηση του αλγορίθμου Dijkstra για τον υπολογισμό της "συντομότερης" διαδρομής μεταξύ δύο κορυφών. Επιστρέφει το κόστος αν υπάρχει διαδρομή, αλλιώς -1. | Ο((Ν+Μ)logN) |
| computeSpanningTree | Εφαρμογή του Prim για τον υπολογισμό ενός Ελάχιστου Δέντρου Κάλυψης (MST). Στο τέλος, επιστρέφει το συνολικό κόστος του δέντρου (sum of weights). Αν ο γράφος είναι συνεκτικός, το δέντρο θα περιλαμβάνει όλες τις κορυφές. Διαφορετικά, υπολογίζει ένα το δέντρο που αφορά όσες κορυφές είναι συνδεδεμένες. | Ο((Ν+Μ)logN) |
| findConnected  Components | Χρησιμοποιούμε την Αναζήτηση Βάθους (DFS) για να υπολογίσουμε πόσες ανεξάρτητες συνιστώσες (δηλαδη σε ποσες ομαδες μπορουμε να τους χωρισουμε) έχει ο γράφος. |  |

**ΕΥΧΑΡΙΣΤΟΥΜΕ**