ΤΕΧΝΙΚΉ ΑΝΑΦΟΡΆ

Δομές Δεδομένων 2021-2022 – 2η Εργασία

Χρησιμοποιήθηκε πίνακας κατακερματισμού ανοιχτής διευθυνσιοδότησης, με **Δευτεροβάθμια Διερεύνηση (Quadratic Probing)**. Ο λόγος που έγινε αυτή η επιλογή, είναι για να χρησιμοποιηθεί το θεώρημα που αναφέρει ότι: Όταν χρησιμοποιείται δευτεροβάθμια διερεύνηση και το μέγεθος του πίνακα είναι πρώτος αριθμός, πάντα μπορεί να εισαχθεί ένα νέο κλειδί, αν ο πίνακας είναι το πολύ κατά το ήμισυ γεμάτος (δηλαδή loadFactor $\leqslant$ 0.5). Έτσι, για να μπορούμε να πούμε εγγυημένα ότι, θα μπορούν να εισαχθούν πάντα τουλάχιστον mx στοιχεία, υλοποιείται πίνακας με μέγεθος, τον πρώτο, πρώτο (που διαιρείται μόνο από το 1 και τον εαυτό του) αριθμό που υπάρχει, αμέσως μετά το διπλάσιο του mx.*

private boolean isPrime(int n):

Η μέθοδος αυτή ελέγχει αν ο δοσμένος αριθμός n είναι πρώτος αριθμός (Διαιρείται μόνο από το 1 και τον εαυτό του). Ελέγχει για κάθε αριθμό από το 2 και μετά, αν διαιρεί τον n, κι αν τον διαιρεί επιστρέφει false (Δεν είναι πρώτος αριθμός). Αυτό γίνεται με ένα for loop, το οποίο εκτελείται έως ότου το j*j γίνει >n, γιατί ο μεγαλύτερος αριθμός που μπορεί να διαιρέσει τον n (και δεν διαιρείται ο ίδιος από προηγούμενούς του αριθμούς [είναι πρώτος]), είναι μικρότερος ή ίσος της τετραγωνικής ρίζας του n. Αν δεν βρεθεί αριθμός που να διαιρεί τον n στην for, επιστρέφει true.

Πολυπλοκότητα:

-Στην βέλτιστη περίπτωση: O(1) (π.χ. n=2). Δεν ξεκινά καν η for. Ο 2 είναι πρώτος αριθμός.

-Στην χειρότερη περίπτωση: O(n^(1/2) -1) (π.χ. n=9 ή n=25). Αφού το for loop εκτελείται έως ότου το j*j να γίνει >n, άρα εκτελείται το πολύ n^(1/2)-1 (μείον ένα λόγω του ότι ξεκινάμε από το 2) φορές.

public WhatAStruct(int mx):

Κατασκευαστής της κλάσης WhatAStruct. Πρώτα βρίσκεται το μέγεθος του πίνακα, που χρειάζεται για να μπορούν να εισαχθούν τουλάχιστον mx στοιχεία (Αν είναι αρνητικό το mx, θετικοποιείται). Σύμφωνα με το Θεώρημα (που αναφέρεται στην αρχή της σελίδας) για τη Δευτεροβάθμια Διερεύνηση, θέλουμε τον πρώτο, πρώτο (που διαιρείται μόνο από το 1 και τον εαυτό του) αριθμό, που υπάρχει αμέσως μετά το διπλάσιο του mx. Άρα, κρατάται σε μια local μεταβλητή size, το διπλάσιο του mx συν 1 (συν 1, για να αποφευχθεί μια επανάληψη, αφού το διπλάσιο ενός αριθμού δεν είναι ποτέ πρώτος [διαιρείται από τον αριθμό εκείνο] – γίνεται μόνο size=mx*2, μόνο αν το mx είναι 1) και μετά με μια while, ελέγχεται σε κάθε επανάληψη, αν η μεταβλητή αυτή είναι πρώτος αριθμός με χρήση της isPrime(size), και την αυξάνουμε κατά 1 αν δεν είναι, μέχρι να γίνει πρώτος. Μόλις βρεθεί το κατάλληλο μέγεθος, δημιουργείται πίνακας Node με μέγεθος size και δημιουργείται ο κόμβος διαγραφής noNode, όπου χαρακτηρίζεται, βάζοντας false, το boolean πεδίο του deleted.

Πολυπλοκότητα:

-Στην βέλτιστη περίπτωση: O(1) (π.χ. n=2). Αν n=2, τότε η isPrime() θα γίνει σε O(1) και η while θα πάρει κατευθείαν μέσα true, άρα O(1)*O(1) = O(1).

-Στην χειρότερη περίπτωση: Οι επαναλήψεις που θα κάνει η while, εξαρτώνται αποκλειστικά από το **mx** και το πόσους αριθμούς μετά το διπλάσιο του, βρίσκεται ο πρώτος, πρώτος αριθμός και έτσι θα γίνουν το πολύ [(n(αριθμός που δόθηκε στην isPrime)^(1/2) -1) * αυτόν τον αριθμό] βήματα. **

private int hash (int id):

Η μέθοδος αυτή βρίσκει το hash για το δοσμένο id. Αν το id<0, το θετικοποιεί, απλά για τον υπολογισμό του hash, αφού δεν υπάρχει αρνητική θέση στον πίνακα. Επιστρέφει το υπόλοιπο της διαίρεσης του id με το μέγεθος του πίνακα της δομής.

Πολυπλοκότητα:

O(1) πάντα. (Εκτέλεση Απλών Εντολών - Δεν υπάρχουν επαναλήψεις)

public boolean insert(Node n):

Η μέθοδος αυτή, προσπαθεί να εισάγει τον κόμβο n, στην δομή this, για την οποία αυτή καλείται. Αρχικά, κρατά στην μεταβλητή hashfirst, το hash για το id του κόμβου που θέλουμε να εισάγουμε. Αρχικοποιείται μια μεταβλητή count με 0, που μετρά τις θέσεις του πίνακα που προσπελάστηκαν και μια μεταβλητή h με το hfirst, η οποία κρατά τη θέση στην οποία γίνονται κάθε φορά οι έλεγχοι (δείκτης). Εκτελείται μια while, η οποία γίνεται μέχρι να βρεθεί κενή θέση στον πίνακα ή να προσπελαστούν όλες οι θέσεις του (ποτέ δεν ξαναπροσπαλαύνει την ίδια θέση). Σε περίπτωση που ο κόμβος που θέλουμε να εισαχθεί είχε πριν διαγραφεί, δηλαδή είναι noNode, τότε γίνεται break από την while και απλά εισάγεται σε αυτό το σημείο ο δοσμένος κόμβος. Σε περίπτωση που κατά την προσπέλαση, πέσει ο δείκτης πάνω σε θέση με κόμβο με ίδιο id με του κόμβου που θέλουμε να εισαχθεί, τότε γίνονται η κατάλληλες ενέργειες, ανάλογα με την τιμή της flag1. Η flag1 είναι μια global μεταβλητή, η οποία δείχνει αν γίνεται κανονική εισαγωγή (flag1=0), αν γίνεται εισαγωγή στοιχείων για την union (flag1=1 – για τα στοιχεία της w WhatAStruct, στην this), ή αν λειτουργεί σύμφωνα με την diff (flag1=2 - για τα στοιχεία της w WhatAStruct, στην this). Αν flag1=0, τότε απλά επιστρέφεται false, γιατί δεν πρέπει να υπάρχουν στην ίδια δομή, κόμβοι με ίδιο id (Το id πρέπει να είναι μοναδικό). Αν flag1=1, τότε μπαίνει για priority στον κόμβο της δομής this, το άθροισμα των priority του κόμβου αυτού και του κόμβου που πήρε σαν όρισμα η insert. Αν flag1=2, τότε μπαίνει για priority στον κόμβο της δομής this, η διαφορά των priority του κόμβου αυτού και του κόμβου που πήρε σαν όρισμα η insert και σε περίπτωση που το τελικό priority βγαίνει αρνητικό, γίνεται θετικό, για να αποκομίσουμε την απόλυτη τιμή του αποτελέσματος, αφού ζητείται αριθμητική διαφορά. Κάθε φορά που, είτε δεν έχει βρεθεί κενή θέση (null ή noNode), είτε δεν έχει βρεθεί κόμβος στην δομή με κοινό id, για να γίνει κάτι από τα παραπάνω, ο μετρητής count αυξάνεται κατά 1 (αν δεν έχει ήδη ξεπεράσει το μέγεθος του πίνακα) και βρίσκεται η επόμενη θέση h που θα προσπελαστεί, ως το υπόλοιπο της διαίρεσης της count (αριθμός κόμβων που έχει προσπελαστεί) στο τετράγωνο, συν το αρχικό hash, hfirst (ουσιαστικά είναι το hash του αθροίσματος αυτού κάθε φορά). Έτσι, είναι σαν να γίνεται έμμεση υλοποίηση της Δευτεροβάθμιας Συνάρτησης επίλυσης συγκρούσεων (F(i) = i^2), μέσα στο h, με την count για i και εξετάζονται μία-μία οι αντίστοιχες θέσεις, καθώς αυξάνεται το count (hfirst, hfirst+1, hfisrt+4, hfirst+9). Σε περίπτωση που βρεθεί χώρος για εισαγωγή στοιχείου που δεν είναι ίδιο με κανένα από αυτά που υπάρχουν ήδη στην δομή και γίνεται εισαγωγή για την union ή κανονική εισαγωγή (flag1!=2), τότε ελέγχεται αν οι θέσεις που προσπελάστηκαν είναι περισσότερες από το μέγεθος του πίνακα. Αυτός ο έλεγχος γίνεται, διότι μπορεί να βρεθεί null κόμβος και να σταματήσει η while, ενώ ταυτόχρονα να έχουν προσπελαστεί ήδη όλες οι θέσεις του πίνακα, άρα θα δείχνει σε λάθος θέση η h, αφού το στοιχείο που πήγε να εισαχθεί δεν χώραγε να μπει στον πίνακα (στην πραγματικότητα αν flag=1, πάντα θα χωράει να μπει κόμβος αφού στην union είχε φτιαχτεί η δομή, έτσι ώστε να χωράει τουλάχιστον τον αριθμό όλων των στοιχείων της this της και της w μαζί – Στην περίπτωση που flag1==2, το να βγει η συνθήκη της

while, count>-this.node.length➔true, σημαίνει ότι δεν βρήκε στοιχείο ίδιο με το δοσμένο στην δομή) και σε αυτήν την περίπτωση απλά επιστρέφεται false. Αλλιώς, εισάγεται ο κόμβος n στην θέση h και επιστρέφεται true.

Οι λόγοι που αποφάσισα να υλοποιηθεί έτσι η insert είναι 2: 1) Για να εκμεταλλευτώ την προσπέλαση που κάνει ήδη η insert στην θέση που αντιστοιχεί κάθε φορά (σύμφωνα με την ακολουθία hfirst-hfirst+1-hfisrt+4-hfirst+9 (1,4,9 κ.λ.π. δίνονται από το count^2), για να γίνουν οι λιγότερες δυνατές προσπελάσεις, αντί να γίνει κανονική σειριακή προσπέλαση όλου του πίνακα κάθε φορά (την ίδια προσπέλαση με την insert κάνει και η contains, αλλά δεν μπορούσα να την εκμεταλλευτώ γιατί δέχεται σαν όρισμα μόνο το id). 2) Για να μην υλοποιηθεί καινούργια μέθοδος, η οποία κατά το ήμισυ θα ήταν ίδια με την insert, ενώ μπορώ απλά να συμπληρώσω λίγο την insert και να χρησιμοποιήσω σαν σηματοδότη για το τι θα κάνει την flag1. Έτσι η insert, εκτός από την εισαγωγή κόμβων σε δομή, χρησιμοποιείται και για την εισαγωγή ανανεωμένων στοιχείων - ψευδοεισαγωγή κόμβου αντικαταστάτη στην δομή και κάνει την λειτουργία που ζητείται, ανεπηρέαστα.

Πολυπλοκότητα:

-Στην βέλτιστη περίπτωση: O(1) (αν δεν συμβεί καμία σύγκρουση).

-Στην χειρότερη περίπτωση: Αν συμβούν συγκρούσεις, ο χρόνος εξαρτάται από το μήκος της ακολουθίας διερευνήσεων, το οποίο εξαρτάται από τα στοιχεία που θα πάνε να εισαχθούν, καθώς και το μέγεθος του πίνακα. Στην χειρότερη περίπτωση, αν γίνουν μόνο συγκρούσεις και το στοιχείο δεν χωρέσει στον πίνακα (και γίνει count=node.length), θα γίνουν node.length επαναλήψεις, άρα θα γίνει η insert σε **O(n)** χρόνο, όπου εδώ n= node.length, [άρα O(this.node.length)].

public Node remove():

Η μέθοδος αυτή απομακρύνει από τη δομή το στοιχείο με την μεγαλύτερη προτεραιότητα*** και επιστρέφει τον αντίστοιχο του κόμβο (αν η δομή είναι κενή επιστρέφει null). Αρχικά, ορίζεται μια boolean μεταβλητή empty που δείχνει αν η δομή είναι άδεια ή όχι (αρχικοποιείται με false) , μια μεταβλητή max που εν τέλει θα έχει το μεγαλύτερο priority στην δομή και μια maxi που θα έχει τη θέση του κόμβου με το μεγαλύτερο priority στον πίνακα. Ο πίνακας αυτός προσπελαύνεται, έως ότου βρεθεί το πρώτη θέση του πίνακα η οποία περιέχει μέσα στοιχείο (δεν έχει null ή noNode). Αν/Όταν βρεθεί αυτό το πρώτο στοιχείο, αρχικοποιούνται οι max και maxi με τα αντίστοιχα priority και τη θέση του (για να μπορούν να ξεκινήσουν μετά οι

συγκρίσεις για να βρεθεί το στοιχείο με το μεγαλύτερο priority) και μπαίνει στην μεταβλητή empty η τιμή false (εφόσον έχει βρεθεί τουλάχιστον ένα στοιχείο στη δομή). Αν η δομή είναι κενή (empty=true), τότε η μέθοδος επιστρέφει απλά null. Αν όμως δεν είναι κενή, τότε προσπελαύνει τον πίνακα, αρχίζοντας από την επόμενη θέση από αυτήν στην οποία βρήκε το πρώτο μη κενό στοιχείο, maxi+1 (αφού όλα τα προηγούμενα θα είναι κενά). Αυτό γίνεται, έτσι ώστε και από τις δύο for, να γίνουν σύνολο this.node.length (μήκος του πίνακα) προσπελάσεις στον πίνακα και να μην ξαναελέγχονται κενές θέσεις. Για κάθε μη κενό στοιχείο που προσπελαύνεται, ελέγχεται αν το priority του είναι μεγαλύτερο του max (του προηγούμενου μεγαλύτερου priority) και αν ναι, τότε το priority αυτό γίνεται το νέο max και η θέση του αντίστοιχου στοιχείου το νέο maxi. Μόλις έχει ελεγχθεί όλη η δομή και έχει βρεθεί επιτυχώς το στοιχείο με την μεγαλύτερη προτεραιότητα, τότε ορίζεται ένας βοηθητικός δείκτης tmp τύπου Node, ο οποίος δείχνει στον κόμβο του πίνακα που βρήκαμε, μετά μπαίνει η αντίστοιχη θέση του πίνακα αυτή να δείχνει στο noNode (απομακρύνεται από τη δόμή) και τέλος επιστρέφεται ο κόμβος tmp.

Πολυπλοκότητα:

(Έστω n=το μέγεθος του πίνακα= this.node.length)

-O(n) πάντα. Αν και η απομάκρυνση του στοιχείου μόνη της γίνεται σε O(1), η εύρεση του στοιχείου με την μεγαλύτερη προτεραιότητα, απαιτεί n προσπελάσεις. Ο λόγος που δεν μπορεί να γίνει προσπέλαση όπως στην insert και την contains,σύμφωνα με την ακολουθία διερευνήσεων, είναι διότι αναζητούμε στοιχείο σύμφωνα με το priority και όχι το id και μάλιστα αυτό το στοιχείο μπορεί να είναι οποιοδήποτε στην δομή. Αλλά λόγω του ότι ξεκινάμε από το maxi στην δεύτερη for, αποφεύγεται να γίνονται έξτρα προσπελάσεις και γίνονται πάντα ακριβώς n.

public boolean contains(int id):

Η μέθοδος αυτή βρίσκει αν υπάρχει στην δομή, κόμβος με id ίδιο με αυτό που παίρνει σαν παράμετρο. Αρχικά, κρατά στην μεταβλητή hashfirst, το hash για το id που θέλουμε να αναζητήσουμε. Αρχικοποιείται μια μεταβλητή count με 0, που μετρά τις θέσεις του πίνακα που προσπελάστηκαν και μια μεταβλητή h με το hfirst, η οποία κρατά τη θέση στην οποία γίνονται κάθε φορά οι έλεγχοι (δείκτης). Εκτελείται μια while, η οποία γίνεται μέχρι να βρεθεί κενή θέση στον πίνακα με null (που σημαίνει ότι δεν έχει εισαχθεί κόμβος με αυτό το id αφού θα έπρεπε να βρίσκεται σε εκείνη τη θέση) ή να βρεθεί κενή θέση στον πίνακα με noNode (που σημαίνει ότι κόμβος με αυτό το id είχε εισαχθεί, απομακρύνθηκε από τη δομή και τώρα δεν έχει ξαναεισαχθεί) ή να προσπελαστούν όλες οι θέσεις του (δεν χώραγε να εισαχθεί από πριν

και δεν μπήκε ποτέ στον πίνακα - ποτέ δεν ξαναπροσπαλαύνει την ίδια θέση). Αν βρεθεί στοιχείο με το ζητούμενο id στην θέση h στην οποία γίνεται κάθε φορά ο έλεγχος, τότε η μέθοδος επιστρέφει true. Κάθε φορά που, δεν έχει συμβεί κάτι από τα παραπάνω, ή δεν έχει βρεθεί κόμβος με το δοσμένο id, ο μετρητής count αυξάνεται κατά 1 (αν δεν έχει ήδη ξεπεράσει το μέγεθος του πίνακα) και βρίσκεται η επόμενη θέση h που θα προσπελαστεί, ως το υπόλοιπο της διαίρεσης της count (αριθμός κόμβων που έχει προσπελαστεί) στο τετράγωνο, συν το αρχικό hash, hfirst (ουσιαστικά είναι το hash του αθροίσματος αυτού κάθε φορά). Έτσι, είναι σαν να γίνεται έμμεση υλοποίηση της Δευτεροβάθμιας Συνάρτησης επίλυσης συγκρούσεων (F(i) = i^2), μέσα στο h, με την count για i και εξετάζονται μία-μία οι αντίστοιχες θέσεις, καθώς αυξάνεται το count (hfirst, hfirst+1, hfisrt+4, hfirst+9). Αν εν τέλει, δεν βρεθεί κόμβος με το δοσμένο id, επιστρέφεται false.

Πολυπλοκότητα:

-Στην βέλτιστη περίπτωση: O(1) (αν δεν συμβεί καμία σύγκρουση).

-Στην χειρότερη περίπτωση: Αν συμβούν συγκρούσεις, ο χρόνος εξαρτάται από το μήκος της ακολουθίας διερευνήσεων, το οποίο εξαρτάται από τα στοιχεία που θα πάνε να εισαχθούν, καθώς και το μέγεθος του πίνακα. Στην χειρότερη περίπτωση, αν γίνουν μόνο συγκρούσεις και το στοιχείο δεν βρεθεί στον πίνακα (και γίνει count=node.length), θα γίνουν node.length επαναλήψεις, άρα θα γίνει η insert σε **O(n)** χρόνο, όπου εδώ n= node.length, [άρα O(this.node.length)].

public WhatAStruct union(WhatAStruct w):

Αρχικά προσπελαύνονται οι πίνακες των this και w, και κρατούνται σε μεταβλητές pl1 και pl2 αντίστοιχα, τα πλήθη των στοιχείων που αυτοί έχουν (όσες θέσεις δεν έχουν null ή noNode). Ύστερα κατασκευάζεται νέο στιγμιότυπο κλάσης WhatAStruct, το str1, με μέγεθος πίνακα κατάλληλο για να χωρέσουν τουλάχιστον pl1+pl2 στοιχεία (παίρνει σαν όρισμα pl1+pl2). Αυτό που ζητείται είναι η επιστρεφόμενη δομή str1 να περιέχει όλα τα στοιχεία των this και w και για τα στοιχεία που αυτές οι δύο έχουν κοινά μεταξύ τους (έχουν ίδιο id), το priority του κόμβου με το αντίστοιχο id που θα βρίσκεται στην str1, να είναι το άθροισμα των priority των δύο κοινών, των this και w. Άρα αυτό που γίνεται είναι ότι, πρώτα γίνεται κανονική εισαγωγή (flag1=0), όλων των στοιχείων της this (νέους κόμβους αντίγραφα αυτών της this) στην str1, μέσω της insert. Μετά το flag1 της str1 γίνεται 1 και γίνεται insert κάθε στοιχείου της w (νέους κόμβους αντίγραφα αυτών της w) στην str1, όπου σύμφωνα με τον τρόπο που είναι υλοποιημένη η insert (όπως εξηγείται παραπάνω), τώρα που flag1 ==1, για όσα στοιχεία της w

υπάρχουν ήδη στην str1 (άρα και στην this), θα αλλάξει το priority τους στην str1, με το άθροισμα των priority αυτουνού και του αντίστοιχου κόμβου της w, ενώ όσα δεν υπάρχουν στην str1 (άρα και στην this), απλά θα γίνει κανονική εισαγωγή στον αντίστοιχο πίνακα αυτής. Τέλος το flag1 της str1 ξαναγίνεται 0, για να γίνεται μετά κανονική εισαγωγή και επιστρέφεται η ζητούμενη δομή str1.

Πολυπλοκότητα:

(Έστω n=το μέγεθος του πίνακα της this = this.node.length

Έστω m=το μέγεθος του πίνακα της w = w.node.length

Έστω p=το μέγεθος του πίνακα της str1 = str1.node.length)

-Στην βέλτιστη περίπτωση: O(n) + O(m) + O(n) * O(1) + O(m) * O(1) = O(n) + O(m)

[O(n), αν n>m - O(m), αν m>n] (αν δεν συμβεί καμία σύγκρουση στις κλήσεις της insert).

-Στην χειρότερη περίπτωση: Αν συμβούν συγκρούσεις στην insert, ο χρόνος εξαρτάται από το μήκος της ακολουθίας διερευνήσεων, το οποίο εξαρτάται από τα στοιχεία που θα πάνε να εισαχθούν, καθώς και το μέγεθος των πινάκων. Στην χειρότερη περίπτωση, αν γίνουν μόνο συγκρούσεις, θα είναι:

O(n) + O(m) + O(n) * O(p-1) + O(m) * O(p-2) = O(n) + O(m) + O((p-1)*n) + O((p-2)*m) ((p-1)*n> n, (p-2)*m>m)= O((p-1)*n) + O((p-2)*m)

[O((p-1)*n), αν (p-1)*n >(p-2)*m - O((p-2)*m), αν (p-2)*m >(p-1)*n].

(p-1 και όχι p, διότι πάντα θα χωράνε τα στοιχεία να εισαχθούν, άρα στην χειρότερη περίπτωση, που βρεθεί διαθέσιμη θέση στην τελευταία θέση, θα έχουν γίνει p-1 επαναλήψεις με την while, ενώ μετά στην χειρότερη περίπτωση, θα βρεθεί διαθέσιμη θέση στην προ-τελευταία θέση [αφού η τελευταία θα είναι κατειλημμένη], άρα θα έχουν γίνει p-2 επαναλήψεις με την while)

public WhatAStruct diff(WhatAStruct w):

Αρχικά προσπελαύνεται ο πίνακας των this, και κρατάται σε μεταβλητή pl1 το πλήθος των στοιχείων που αυτή έχει (όσες θέσεις δεν έχουν null ή noNode). Ύστερα κατασκευάζεται νέο στιγμιότυπο κλάσης WhatAStruct, το str2, με μέγεθος πίνακα κατάλληλο για να χωρέσουν τουλάχιστον pl1 στοιχεία (παίρνει σαν όρισμα pl1 – θα περιέχει ακριβώς όσα στοιχεία

περιέχει και η this). Αυτό που ζητείται στην ουσία, είναι η επιστρεφόμενη δομή str2 να περιέχει όλα τα στοιχεία της this και για τα στοιχεία που οι this και w έχουν κοινά μεταξύ τους (έχουν ίδιο id), το priority του κόμβου με το αντίστοιχο id που θα βρίσκεται στην str2, να είναι η διαφορά των priority των δύο κοινών, των this και w. Άρα αυτό που γίνεται είναι ότι, πρώτα γίνεται κανονική εισαγωγή (flag1=0), όλων των στοιχείων (νέους κόμβους αντίγραφα αυτών της this) της this στην str2, μέσω της insert. Μετά το flag1 της str1 γίνεται 2 και γίνεται insert κάθε στοιχείου της w (νέους κόμβους αντίγραφα αυτών της w) στην str1, όπου σύμφωνα με τον τρόπο που είναι υλοποιημένη η insert (όπως εξηγείται παραπάνω), τώρα που flag1 ==2, για όσα στοιχεία της w υπάρχουν ήδη στην str1 (άρα και στην this), θα αλλάξει το priority τους στην str1, με τη διαφορά των priority αυτουνού και του αντίστοιχου κόμβου της w, ενώ όσα δεν υπάρχουν στην str1 (άρα και στην this), απλά θα αγνοηθούν και η insert θα επιστρέψει false (άκυρο). Τέλος το flag1 της str2 ξαναγίνεται 0, για να γίνεται μετά κανονική εισαγωγή και επιστρέφεται η ζητούμενη δομή str2.

<u>Πολυπλοκότητα:</u>

(Έστω n=το μέγεθος του πίνακα της this = this.node.length

Έστω m=το μέγεθος του πίνακα της w = w.node.length

Έστω p=το μέγεθος του πίνακα της str2 = str2.node.length)

-Στην βέλτιστη περίπτωση: O(n) + O(m) + O(n) * O(1) + O(m) * O(1) = <u>O(n) + O(m)</u>

[O(n), αν n>m - O(m), αν m>n] (αν δεν συμβεί καμία σύγκρουση στις κλήσεις της insert).

-Στην χειρότερη περίπτωση: Αν συμβούν συγκρούσεις στην insert, ο χρόνος εξαρτάται από το μήκος της ακολουθίας διερευνήσεων, το οποίο εξαρτάται από τα στοιχεία που θα πάνε να εισαχθούν, καθώς και το μέγεθος των πινάκων. Στην χειρότερη περίπτωση, αν γίνουν μόνο συγκρούσεις, θα είναι:

O(n) + O(m) + O(n) * O(p-1) + O(m) * O(p-2) = O(n) + O(m) + O((p-1)*n) + O((p-2)*m) ((p-1)*n> n, (p-2)*m>m)= <u>O((p-1)*n) + O((p-2)*m)</u>

[O((p-1)*n), αν (p-1)*n >(p-2)*m - O((p-2)*m), αν (p-2)*m >(p-1)*n].

(p-1 και όχι p, διότι πάντα θα χωράνε τα στοιχεία να εισαχθούν, άρα στην χειρότερη περίπτωση, που βρεθεί διαθέσιμη θέση στην τελευταία θέση, θα έχουν γίνει p-1 επαναλήψεις με την while, ενώ μετά στην χειρότερη περίπτωση, θα βρεθεί κενή θέση (άρα δεν υπάρχει το στοιχείο στον πίνακα) στην προ-τελευταία θέση [αφού η τελευταία θα είναι κατειλημμένη], άρα θα έχουν γίνει p-2 επαναλήψεις με την while).

public WhatAStruct kbest(int k):

Αρχικά κατασκευάζεται βοηθητικό WhatAStruct str3, το οποίο παίρνει σαν όρισμα το μισό του μεγέθους του πίνακα της this (άρα η str3 θα έχει πίνακα με μέγεθος, τουλάχιστον εκείνου του πίνακα της this). Ύστερα, προσπελαύνεται η this και κάθε στοιχείο που έχει μη κενό εισάγεται στην str3 (νέοι κόμβοι αντίγραφα αυτών της this), ενώ ταυτόχρονα υπολογίζεται στο πλήθος pl1 που κρατά το πλήθος των στοιχείων της this. Εφόσον ζητείται δομή με τα k μεγαλύτερα σε priority στοιχεία της this, αν το k>pl1, τότε γίνεται το k=pl1 (η επιστρεφόμενη δομή θα έχει όλα τα στοιχεία της this) και ύστερα, όποιο κι αν είναι το k υλοποιείται καινούργιά δομή str3 (αυτή που θα επιστραφεί) με όρισμα το k (θα χωρά τουλάχιστον k στοιχεία). Μετά για k επαναλήψεις, βρίσκεται το μεγαλύτερο σε priority στοιχείο της str3 (άρα και της this), απομακρύνεται από αυτήν και αποθηκεύεται ο επιστρεφόμενος κόμβος σε μια βοηθητική tmp, αντίγραφο της οποίας (νέος κόμβος) μετά εισάγεται στην str4. Η διαδικασία μετά επαναλαμβάνεται για την πλέον ανανεωμένη str3, για να βρεθεί ο κόμβος με το επόμενο μεγαλύτερο priority. Τέλος επιστρέφεται η ζητούμενη δομή str4.

Πολυπλοκότητα:

(Έστω n=το μέγεθος του πίνακα της this = this.node.length

Έστω m=το μέγεθος του πίνακα της str3 = str3.node.length (>=this.node.length)

Έστω p=το μέγεθος του πίνακα της str4 = str3.node.length)

-Στην βέλτιστη περίπτωση: O(n)*O(1) + O(k)* (O(m)+O(1)) = O(n) + O(k*m) + O(k) (m>=n) **= O(k*m)** (αν δεν συμβεί καμία σύγκρουση σε καμία insert - μπαίνει O(k) και όχι, γιατί μπορούν να συμβούν και λιγότερες από το δοσμένο k επαναλήψεις στην δεύτερη for).

-Στην χειρότερη περίπτωση: Αν συμβούν συγκρούσεις στις insert, ο χρόνος εξαρτάται από το μήκος της ακολουθίας διερευνήσεων, το οποίο εξαρτάται από τα στοιχεία που θα πάνε να εισαχθούν, καθώς και το μέγεθος του πίνακα. Στην χειρότερη περίπτωση, αν γίνουν μόνο συγκρούσεις θα γίνουν: **O(n)*O(m) + O(k)* (O(m)+O(p)) = O(n*m) + O(k*m) + O(k*p)** (k<=p<=n<=m) **= O(n*m) + O(k*m) = O(n*m)**

Παράδειγμα (με διάφορες εισαγωγές κόμβων), με σκοπό την παρουσίαση της οργάνωσης των στοιχείων στην δομή:

mx=5
size=11 (first prime number after double of mx) hash = id % 11

Δευτεροβάθμια Διερεύνηση: F(i) = i^2, (h(k) + i^2) mod tableSize, i = 0, 1, 2, . . .
(Βήματα Ακολουθίας: 0, 1,4,9,16…)

							π.χ. ↓			
Insert			n1	n2	n3	n4	n5	n6	n7	n8
Node(id,priority) [in order==>]:			(11,32)	(89,12)	(34,13)	(60,98)	(59,57)	(67,45)	(71,44)	(93,33)
	n6 (first try)	n6 (second)			n6 (third)					n6 (placed)
	↓	↓			↓					↓
0	1	2	3	4	5	6	7	8	9	10
(11,32)	(89,12)	(34,13)		(59,57)	(60,98)	(71,44)			(93,33)	(67,45)

(Τα κενά μπορούν να είναι null ή και noNode)

*με αυτόν τον τρόπο και η Γραμμική Διερεύνηση θα δούλευε, απλά προτιμήθηκε προσωπικά η Δευτεροβάθμια Διερεύνηση, για λόγους Αντιμετώπισης Πρωτογενούς Δημιουργίας Δεσμών)

**Η μεγαλύτερη καταγεγραμμένη απόσταση μεταξύ πρώτων αριθμών, είναι ένας αριθμός με μήκος 1113106 και αξία 25.90. Άρα, θεωρητικά, στην χειρότερη περίπτωση θα γίνουν επαναλήψεις όσο: (αυτός ο αριθμός, μείον 1 [λόγω του ότι αρχίζουμε με 2*mx+1]) επί (n^(1/2) -1).

***(το πρώτο που θα συναντήσει με αυτήν την προτεραιότητα)

-ΣΩΤΗΡΙΟΣ ΔΗΜΗΤΡΑΚΟΥΛΑΚΟΣ – Ε20040