Γεώργιος Γιαλούρης
2019030063

A1. Δημιουργώ μια κλάση Node, η οποία περιέχει τους ακεραίους x και y και έναν δείκτη στο επόμενο node. Επίσης, δημιουργώ μια κλάση List η οποία περιέχει δυο δείκτες, τον head και τον tail. Για την insert, αφού κάνω τους απαραίτητους ελέγχους ,ώστε να μην έχω στοιχεία εκτός ορίων ή επαναλαμβανόμενα ζεύγη, δημιουργώ ένα καινούριο node, το οποίο είτε το εισάγω στην αρχή της λίστας, αν αυτή είναι άδεια (head = null), είτε στο τέλος της λίστας χρησιμοποιώντας το tail (για να αποφύγω την διάσχιση της λίστας) και αλλάζοντας το next του προηγούμενου node να δείχνει αντί για null το καινούριο node. Για την search χρησιμοποιώ ένα current node και διασχίζω την λίστα με τα nodes με την βοήθεια μιας for και της getNext. Άμα βρεθεί σημείο με ίδιο x και y, τότε επιστέφω true, αλλιώς false. Επίσης, στην search μετράω με τον multicounter(1) τις συγκρίσεις που εξαρτώνται από τον αριθμό των σημείων, δηλαδή την current != null , την current.getNext και τις συγκρίσεις των x και y προσέχοντας όμως να μετριέται μόνο η μια σύγκριση σε περίπτωση που το x προκύψει διαφορετικό.

A2. Δημιουργώ μια κλάση Hash, η οποία περιέχει έναν πίνακα από List (υλοποιημένη στο A1) μεγέθους M = 100 (για την καλύτερη σύγκριση των δομών αργότερα). Στον constructor αυτής της κλάσης αρχικοποιώ τον πίνακα και τις λίστες σε κάθε θέση του πίνακα με μια for.
Για την insert φτιάχνω πρώτα την μέθοδο getHash, ώστε να υπολογίζω κάθε φορά σε ποια θέση του πίνακα θα μπει το σημείο και στη συνέχεια το εισάγω στο τέλος αυτής της λίστας με την insert του A1. Για την search, υπολογίζω το Hash με τον ίδιο τρόπο και στη συνέχεια κάνω αναζήτηση στη συγκεκριμένη λίστα με την search του A1. Η Hash και η List έχουν παρόμοια υλοποίηση. Επομένως, φτάνει να μετρήσω τις συγκρίσεις που εξαρτώνται από τον αριθμό των σημείων για να βγάλω συμπέρασμα ως προς την απόδοση των δυο.

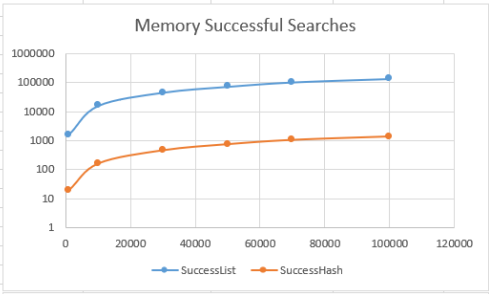
B1. Δημιουργώ μια κλάση DataPage, η οποία περιέχει έναν πίνακα data από ακεραίους (για τα διαδοχικά ζεύγη x και y), το numberOfPoints (για να ξέρω πόσα σημεία έχει μέσα το DataPage), το position (δηλαδή η θέση του DataPage στο αρχείο), το nextPage (η θέση της επόμενης σελίδας της λίστας στο αρχείο, χρήσιμο για το B2) και το PAGE_SIZE (το οποίο έχει δοθεί 256Bytes). Το nextPage ορίστηκε int αντί για long, διότι αν ήταν long (8 bytes) μαζί με το numberOfPoints (4 bytes) θα έπιαναν 12 bytes στη σελίδα και άρα θα έμενε χώρος για 30 ζεύγη x,y και θα περίσσευαν 4 bytes που δεν θα μπορούσαμε να τα αξιοποιήσουμε. Ενώ, τώρα χωράνε ακριβώς 31 ζεύγη x,y μαζί με τα numberOfPoints και nextPage. Επίσης, έχουμε περιορισμένο αριθμό δεδομένων και άρα δεν θα μας δημιουργήσει πρόβλημα ο int.
Στη συνέχεια, δημιουργώ μια κλάση DataPageHandling, στην οποία φτιάχνω τις createFile, openFile και closeFile για την διαχείριση του αρχείου. Μετά, φτιάχνω την writePage, στην οποία παίρνω ως όρισμα ένα page. Δημιουργώ ένα bytearrayoutputstream για να γράψω στο αρχείο και ένα wrapper dataoutputstream για να γράψω τα δεδομένα της σελίδας ως integers. Μετατρέπω

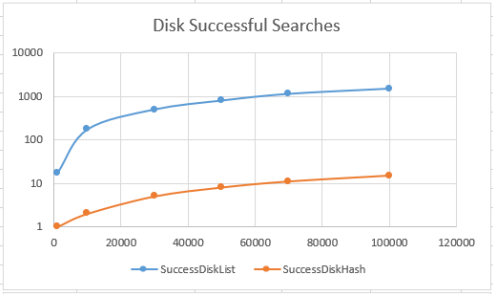
τα δεδομένα σε array από bytes και μετα τα μεταφέρω σε έναν buffer μεγέθους datapage. Τέλος, χρησιμοποιώ την seek για να πάω στην κατάλληλη θέση του αρχείου και την write για να γράψω το page. Έπειτα, φτιάχνω την readPage, στην οποία παίρνει ως όρισμα μια θέση στο αρχείο και επιστρέφει την αντίστοιχη σελίδα. Με την seek και την read πάω και διαβάζω την σελίδα(256bytes). Χρησιμοποιώ bytearrayinputstream και datainputstream για να διαβάζω integers. Διαβάζω τους πρώτους 2 integers (numberOfPoints και nextPage) και μετα με μια for παίρνω τα ζεύγη x,y. Τέλος, δημιουργώ ένα intstance data page με τα παραπάνω στοιχεία στον constructor και επιστρέφω αυτό το page.
Δημιουργώ μια κλάση DiskList, η οποία περιέχει 2 ints το firstPage και το lastPage, τα οποία αναφέρονται στις θέσεις της πρώτης και της τελευταίας σελίδας της λίστας. Τα αρχικοποιω σε -1 για να ξέρω πότε η λίστα είναι άδεια. Έπειτα, φτιάχνω την insert, η οποία έχει 3 περιπτώσεις: α) η λίστα να είναι άδεια, β) η τελευταία σελίδα της λίστας να μην έχει γεμίσει (< 31 ζεύγη) και γ) η τελευταία σελίδα της λίστας να έχει γεμίσει. Στην πρώτη περίπτωση δημιουργούμε ένα datapage με το καινούριο στοιχείο και με numofpoints = 1 και το προσθέτουμε το καινούριο στο τέλος του αρχείου και θέτουμε τα first και last page ίσα με αυτή τη θέση. Στην δεύτερη περίπτωση, προσθέτουμε το στοιχείο στην τελευταία σελίδα και αυξάνουμε το numofpoints κατά 1. Στην τρίτη περίπτωση, ενημερώνω το nextPage της τελευταία σελίδας (= raf.length) και μετα φτιάχνω και γράφω στο τέλος του αρχείου ένα καινούριο datapage στο οποίο έχω προσθέσει το νέο σημείο.
Τέλος, για την search διαβάζω απτό αρχείο και διασχίζω όλες τις σελίδες της λίστας χρησιμοποιώντας το nextPage που έχω αποθηκευμένο σε κάθε page μέχρι να βρω ένα στοιχείο σε κάποιο απτά data pages που να είναι ίδιο με αυτό που αναζητώ. Επιπλέον, κάθε φορά που διαβάζω μια σελίδα αυξάνω τον multicounter(2).

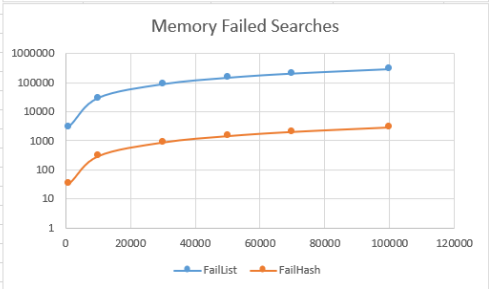
B2. Δημιουργώ την κλάση DiskHash, η οποία λειτουργεί όμοια με την Hash με την διαφορά ότι έχει πίνακα από DiskList αντί για List.

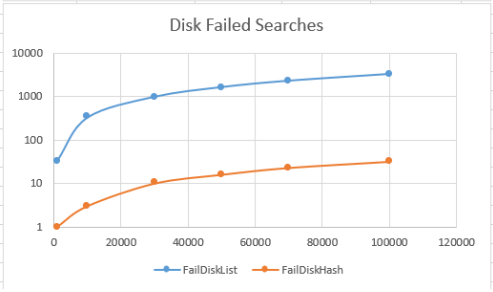
Γ.

K	SuccessList	SuccessHash	K	SuccessDiskList	SuccessDiskHash	K	FailList	FailHash	K	FailDiskList	FailDiskHash
1000	1629	20	1000	17	1	1000	3001	33	1000	33	1
10000	15969	162	10000	172	2	10000	30001	302	10000	323	3
30000	45627	459	30000	491	5	30000	90001	900	30000	968	10
50000	72876	741	50000	784	8	50000	150001	1496	50000	1613	16
70000	103413	1040	70000	1112	11	70000	210001	2094	70000	2259	23
100000	136313	1364	100000	1466	15	100000	300001	2997	100000	3226	32









Η δομή hash είναι πιο αποδοτική από την list για την μέθοδο search, όπως φαίνεται και από τις μετρήσεις, διότι τα δεδομένα μοιράζονται σε M μικρότερες λίστες και με την getHash μας είναι εύκολο να βρούμε σε ποια από αυτές βρίσκεται το σημείο που θέλουμε. Ενώ στην list διασχίζουμε μια μεγάλη λίστα με όλα τα στοιχεία. Αυτό γίνεται πιο ξεκάθαρο στις μετρήσεις όσο αυξάνεται ο αριθμός των στοιχείων.

```
------------------------------ K = 1000 ------------------------------
Average number of comparisons for successful List searches: 1629
Average number of comparisons for failed List searches: 3001
Average number of comparisons for successful Hash searches: 20
Average number of comparisons for failed Hash searches: 33
Average number of comparisons for successful DiskList searches: 17
Average number of comparisons for failed DiskList searches: 33
Average number of comparisons for successful DiskHash searches: 1
Average number of comparisons for failed DiskHash searches: 1
------------------------------ K = 10000 ------------------------------
Average number of comparisons for successful List searches: 15969
Average number of comparisons for failed List searches: 30001
Average number of comparisons for successful Hash searches: 162
Average number of comparisons for failed Hash searches: 302
Average number of comparisons for successful DiskList searches: 172
Average number of comparisons for failed DiskList searches: 323
Average number of comparisons for successful DiskHash searches: 2
Average number of comparisons for failed DiskHash searches: 3
------------------------------ K = 30000 ------------------------------
Average number of comparisons for successful List searches: 45627
Average number of comparisons for failed List searches: 90001
Average number of comparisons for successful Hash searches: 459
Average number of comparisons for failed Hash searches: 900
Average number of comparisons for successful DiskList searches: 491
Average number of comparisons for failed DiskList searches: 968
Average number of comparisons for successful DiskHash searches: 5
Average number of comparisons for failed DiskHash searches: 10
------------------------------ K = 50000 ------------------------------
Average number of comparisons for successful List searches: 72876
Average number of comparisons for failed List searches: 150001
Average number of comparisons for successful Hash searches: 741
Average number of comparisons for failed Hash searches: 1496
Average number of comparisons for successful DiskList searches: 784
Average number of comparisons for failed DiskList searches: 1613
Average number of comparisons for successful DiskHash searches: 8
Average number of comparisons for failed DiskHash searches: 16
------------------------------ K = 70000 ------------------------------
Average number of comparisons for successful List searches: 103413
Average number of comparisons for failed List searches: 210001
Average number of comparisons for successful Hash searches: 1040
Average number of comparisons for failed Hash searches: 2094
Average number of comparisons for successful DiskList searches: 1112
Average number of comparisons for failed DiskList searches: 2259
Average number of comparisons for successful DiskHash searches: 11
Average number of comparisons for failed DiskHash searches: 23
------------------------------ K = 100000 ------------------------------
Average number of comparisons for successful List searches: 136313
Average number of comparisons for failed List searches: 300001
Average number of comparisons for successful Hash searches: 1364
Average number of comparisons for failed Hash searches: 2997
Average number of comparisons for successful DiskList searches: 1466
Average number of comparisons for failed DiskList searches: 3226
Average number of comparisons for successful DiskHash searches: 15
Average number of comparisons for failed DiskHash searches: 32
```