HW 231 Labs Report , Marinos Antoniou & Constantinos Theophilou Team 43

Πρώτού ξεκινήσουμε με την παρουσίαση των γραφικών , θεωρούμε σημαντικό να εξηγήσουμε πώς έχουν παρθεί η κάθε μετρήσεις για κάθε κλάση.

Μέθοδος Υπολογισμού μνήμης για Trie με χρήση στατικού πίνακα :

```
public int calcMem() {
    return calcMem(root);
}

private int calcMem(TrieNode node) {
    if (node == null) {
        return 0;
    }

    int memory = 26 * 8; // Size of children array (26 pointers, assuming 4 bytes each)
    memory += 4 + 4 + 4; // Sizes of wordLength, importance, and other metadata
    for (TrieNode child : node.children) {
        memory += calcMem(child);
    }

    return memory;
}
```

Η μνήμη υπολογίζεται προσθέτοντας 104 bytes για τον πίνακα παιδιών (26 pointers), 12 bytes για τις μεταβλητές του κόμβου (wordLength, importance, metadata), και την αναδρομική μνήμη των παιδιών του κόμβου. Προστίθενται +4 bytes για δίκαιη σύγκριση με Robin Hood hashing.

Μέθοδος Υπολογισμού μνήμης για Trie με χρήση RobinHood πίνακα:

```java
public int calcMem() {
    return calcMem(root);
}

private int calcMem(TrieNode node) {
    if (node == null) {
        return 0;
    }

    // Memory for the current TrieNode:
    int memory = 4; // wordLength (4 bytes)

    // Add memory used by the Robin Hood Hashing structure
    memory += calcRobinHoodMem(node.children);

    // Recursively calculate memory for child nodes
    for (char c = 'a'; c <= 'z'; c++) {
        TrieNode child = node.children.search(c);
        if (child != null) {
            memory += calcMem(child);
        }
    }

    return memory;
}
```

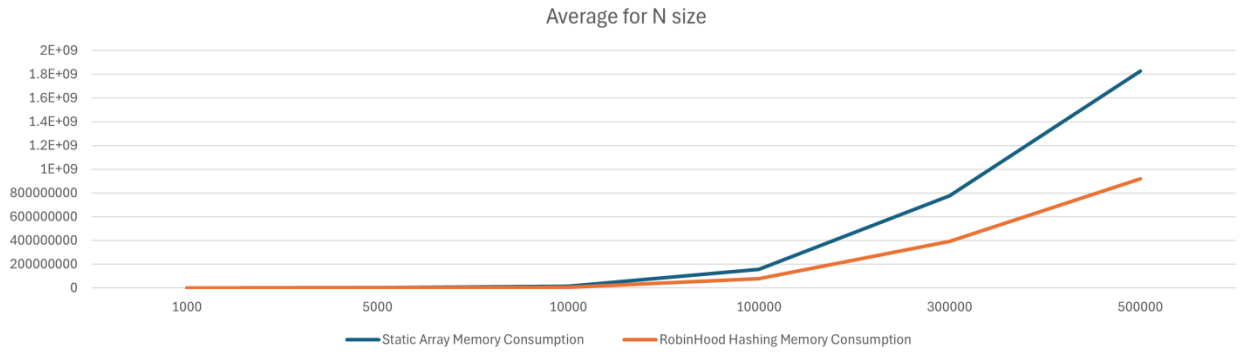
```java
private int calcRobinHoodMem(RobinHoodHashing robinHoodHashing) {
    if (robinHoodHashing == null || robinHoodHashing.table == null) {
        return 0;
    }

    int memory = 0;

    // Memory for the hash table and metadata
    memory += robinHoodHashing.capacity * 12; // Each entry: 1 char (2 bytes) + 1 TrieNode (8 bytes) + probeLength
                                              // (4 bytes)
    memory += 12; // RobinHoodHashing metadata (capacity, size, maxProbeLength)

    return memory;
}
```

Η μνήμη στο TrieHashing υπολογίζεται προσθέτοντας 4 bytes για κάθε κόμβο (wordLength), τη μνήμη της δομής Robin Hood Hashing (12 bytes ανά θέση του πίνακα, συν 12 bytes για μεταδεδομένα), και τη μνήμη όλων των παιδιών του κόμβου. Η συνάρτηση calcMem καλεί αναδρομικά κάθε παιδί για να υπολογίσει τη συνολική μνήμη της δομής, περιλαμβάνοντας τα κλειδιά, τους δείκτες στους κόμβους και τις αποστάσεις probe.



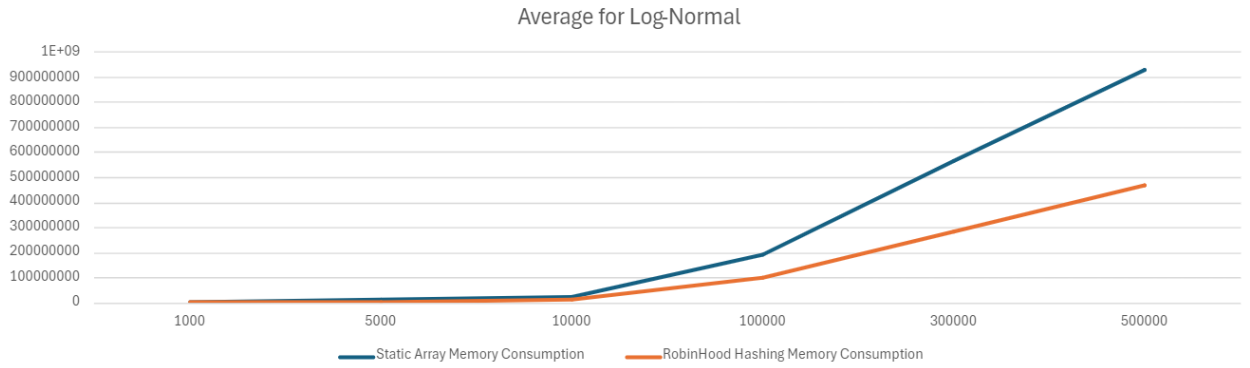
SIZE N TESTS	Test 1		Test 2		Test 3	
Word Count (and size N)	Static Array Memory Consumptio	RobinHood Hashing Memory Consumptio	Static Array Memory Consumptio	RobinHood Hashing Memory Consumptio	Static Array Memory Consumptio	RobinHood Hashing Memory Consumptio
1000 (N = 3)	335840	177524	338140	178976	334180	176690
5000 (N = 5)	3307040	1709954	3297800	1703300	3307920	1711780
10000 (N = 8)	12787280	6520150	12807740	6534138	12797180	6527276
100000 (N = 10)	155694220	78820278	155672000	78806278	155657920	78801702
300000 (N = 15)	777536320	392519164	777586700	392550712	777588240	392546706
500000 (N = 20)	1828519660	919383068	1828465760	919329658	1828424180	919323052

Test 4		Test5		Test Averages		Ratio
Static Array Memory Consumptio	RobinHood Hashing Memory Consumptio	Static Array Memory Consumptio	RobinHood Hashing Memory Consumptio	Static Array Memory Consumption	RobinHood Hashing Memory Consumption	Static / Robinhood
335060	176956	333740	176584	335412	177346	1.891285961
3297360	1704026	3301760	1706386	3302376	1707089.2	1.934506998
12811920	6534966	12815660	6536680	12803956	6530642	1.96059683
155633940	78789914	155663860	78808074	155664388	78805249.2	1.975304813
777557880	392528884	777554140	392532252	777564656	392535543.6	1.980877066
1828424620	919323558	1828494360	919366178	1828465716	919345102.8	1.988878508

Η γραφική παράσταση συγκρίνει τη μνήμη που καταναλώνεται από το Static Array (μπλε γραμμή) και το Robin Hood Hashing (πορτοκαλί γραμμή) καθώς αυξάνεται ο αριθμός των λέξεων και το μέγεθος . Παρατηρούμε ότι το Static Array καταναλώνει σημαντικά περισσότερη μνήμη, με τη διαφορά να αυξάνεται εκθετικά καθώς μεγαλώνει το N. Οι πίνακες παρουσιάζουν τα αποτελέσματα από πειράματα (Tests 1-5) για διαφορετικά N και τον μέσο όρο κατανάλωσης μνήμης, όπου το Static Array είναι σταθερά σχεδόν διπλάσιας κατανάλωσης σε σχέση με το Robin Hood Hashing. Το πηλίκο "Static / Robin Hood" κυμαίνεται από 1.89 έως 1.99, δείχνοντας την υψηλή αποδοτικότητα του Robin Hood Hashing. Η πορτοκαλί γραμμή παραμένει πολύ πιο χαμηλή, αποδεικνύοντας ότι το Robin Hood Hashing διαχειρίζεται τη μνήμη πολύ πιο αποδοτικά, ιδιαίτερα για μεγάλα σύνολα δεδομένων. Συμπερασματικά, το Robin Hood Hashing είναι η βέλτιστη επιλογή για εφαρμογές που απαιτούν αποτελεσματική χρήση της μνήμης.

Αναφορικά με το πείραμα που πρέπει να χρησιμοποιήσουμε κατανομή :

Στο πείραμα χρησιμοποιήθηκε η log-normal κατανομή για την παραγωγή λέξεων με μήκη που αντικατοπτρίζουν ρεαλιστικές γλωσσικές δομές. Σύμφωνα με την έρευνα "Efficient Generation of Random Sequences" (https://arxiv.org/pdf/1709.01712), ιδιαίτερα στις σελίδες 4-6, η log-normal κατανομή παράγει κυρίως μικρές τιμές με ορισμένες μεγάλες αποκλίσεις, γεγονός που την καθιστά κατάλληλη για μήκη λέξεων. Οι παράμετροι της κατανομής καθορίστηκαν με βάση τη μελέτη "On the Log-Normality of Natural Language Phenomena" (https://arxiv.org/pdf/1207.2334), στις σελίδες 2-3, όπου η χρήση του μ (μέση τιμή του log) και σ (διακύμανση) αναλύεται για δεδομένα γλώσσας. Οι τιμές μ_min=1.5, μ_max=3.0, σ_min=0.3, σ_max=0.7 επιλέχθηκαν τυχαία σε κάθε δοκιμή, εξασφαλίζοντας μέσο μήκος λέξεων 3-30 χαρακτήρες με ρεαλιστική διακύμανση, έτσι ώστε να εμφανίζονται κατανεμημένα σε λογικά μεγέθη οι λέξεις, με τις περισσότερες που παράγονται να είναι μεταξύ 8-12 χαρακτήρων.



LOG-NORMAL DISTRIBUTION TESTS	Test 1		Test 2		Test 3	
Word Count	Static Array Memory Consumptio	RobinHood Hashing Memory Consumptio	Static Array Memory Consumptio	RobinHood Hashing Memory Consumptio	Static Array Memory Consumptio	RobinHood Hashing Memory Consumptio
1000	2304940	1162632	2382160	1200890	2196480	1107666
5000	10939720	5525892	10555380	5334060	10584640	5347154
10000	20576820	10416864	21059940	10659584	20940040	10597760
100000	194520920	98024786	194685700	98105726	193736840	97631590
300000	565252820	285965820	565051960	285860350	568299380	287496520
500000	930173860	469699456	929567760	469385888	930812520	470023520

Test 4		Test5		Test Averages		Ratio
Static Array Memory Consumptio	RobinHood Hashing Memory Consumptio	Static Array Memory Consumptio	RobinHood Hashing Memory Consumptio	Static Array Memory Consumption	RobinHood Hashing Memory Consumption	Static / Robinhood
2212760	1116030	2236080	1128010	2266484	1143046	1.982846529
10805080	5458974	10646240	5378690	10706212	5408954	1.979349797
21194360	10726954	21072040	10663974	20968640	10613027	1.976745431
194890960	98195108	194810000	98162894	194528884	98024021	1.984502191
565555320	286109916	567132280	286915840	566258352	286469689	1.976678068
932842680	471025892	930068920	469635602	930693148	469954072	1.980391711

Η γραφική παράσταση συγκρίνει τη μνήμη που καταναλώνουν οι δομές Static Array (μπλε γραμμή) και Robin Hood Hashing (πορτοκαλί γραμμή) κατά την εφαρμογή της log-normal κατανομής για τη δημιουργία δεδομένων. Τα αποτελέσματα δείχνουν ότι το Static Array καταναλώνει σταθερά περισσότερη μνήμη από το Robin Hood Hashing, με τη διαφορά να αυξάνεται όσο μεγαλώνει ο αριθμός των λέξεων. Οι πίνακες αποκαλύπτουν ότι το πηλίκο "Static / Robin Hood" κυμαίνεται μεταξύ 1.97 και 1.98, αποδεικνύοντας ότι το Robin Hood Hashing είναι σχεδόν διπλάσια πιο αποδοτικό. Το Static Array εμφανίζει υψηλή κατανάλωση μνήμης, ιδιαίτερα για μεγαλύτερα σύνολα δεδομένων (500.000 λέξεις), ενώ το

Robin Hood Hashing κλιμακώνεται πιο ομαλά. Τα δεδομένα επιβεβαιώνουν τη σημαντική εξοικονόμηση μνήμης που προσφέρει το Robin Hood Hashing, καθιστώντας το ιδανική επιλογή για μεγάλης κλίμακας εφαρμογές αποθήκευσης δεδομένων. Συνολικά, το πείραμα υπογραμμίζει την αποδοτικότητα της δομής Robin Hood Hashing για συστήματα με αυστηρούς περιορισμούς μνήμης.