

**ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ**

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΡΓΑΣΤΗΡΙΟ ΗΛΕΚΤΡΟΝΙΚΗΣ

**Σχεδιασμός Ενσωματωμένων Συστημάτων**

9ο εξάμηνο

2η Εργαστηριακή Άσκηση:

**Ασκήσεις στη Βελτιστοποίηση Δυναμικών Δομών Δεδομένων (Dynamic Data Type Refinement – DDTR)**

03114681: Maliganis Nikolaos

# Εισαγωγή

Σκοπός της εργαστηριακής αυτής άσκησης είναι η βελτιστοποίηση δυναμικών δομών δεδομένων δύο διαδικτυακών εφαρμογών, του Deficit Round Robin (DRR) και του αλγόριθμου Dijkstra με τη χρήση της μεθοδολογίας «Βελτιστοποίηση Δυναμικών Δομών Δεδομένων» - DDTR (Dynamic Data Type Refinement). Οι δυναμικές δομές δεδομένων των **παραπάνω** αλγορίθμων, θα βελτιστοποιηθούν ως προς :

* **Memory access** (την πρόσβαση στη μνήμη αναφορικά με την προσπέλαση δεδομένων)
* **Memory footprint** (τη μέγιστη μνήμη που καταλαμβάνουν κατά την διαχείριση των δεδομένων)

Για την υλοποίηση της εργαστηριακής άσκησης χρησιμοποιήθηκαν:

* **Mac OS X** (10.14.6)
* gcc for Mac-os - clang version 11.0.0 (clang-1100.0.33.17)
* **valgrind-3.16.1** (Massif, Lackey Valgrind Suite)

# Deficit Round Robin

Αναφορικά με την διαδικασία που ακολουθήθηκε για τον DRR, η υλοποίηση του κώδικα βρίσκεται στα αρχεία drr.h και drr.c χρησιμοποιώντας και το Lib για DDTR. Η μεταγλώττιση θα λάβει χώρα 9 φορές επιλέγοντας όλους τους δυνατούς συνδυασμούς για τις 3 διαφορετικές δομές δεδομένων που θα πρέπει να ελεγχθούν (Single-Linked-List (SLL), Double-Linked-List (DLL) και Dynamic Array (DA)). Αυτό πραγματοποιείται «σχολιάζοντας» κάθε φορά το definition που θέλουμε να συμπεριλάβουμε κάθε φορά στη συγκεκριμένη μεταγλώττιση.

Για κάθε ένα από τα 9 εκτελέσιμα, θε πρέπει να παραχθεί το Memory Access και το Memory Footprint χρησιμοποιώντας το valgrind tool, όπως παρουσιάζεται παρακάτω:

* $ valgrind --log-file="mem\_access.txt" --tool=lackey --trace-mem=yes ./drri
* $ valgrind --tool=massif ./drri

Στον παρακάτω πίνακα 1., παρουσιάζονται όλα τα αποτελέσματα και στις 9 περιπτώσεις:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | Client List | Packet List | Memory Access | Memory Footprint | | SLL | SLL | 74843453 | 838.9 KB | | SLL | DLL | 79666891 | 1091.4 KB | | SLL | DA | 459437498 | 1.522 MB | | DLL | SLL | 77744776 | 934.1 KB | | DLL | DLL | 78456123 | 1094.4 KB | | DLL | DA | 456943681 | 1.539 MB | | DA | SLL | 79257559 | 821.3 KB | | DA | DLL | 79923684 | 1039.6 KB | | DA | DA | 463461777 | 1.586 MB | |
| Πίνακας 1. Mem-Accesses και Mem-Footprint για τους συνδυασμούς DRR |

Ακολουθεί το διάγραμμα Memory Accesses και Memory Footprint για όλους τους δυνατούς συνδυασμούς δυναμικών δομών δεδομένων στο DDR.

|  |
| --- |
| Chart  Description automatically generated |
| Διάγραμμα 1. Mem-Accesses και Mem-Footprint για τους συνδυασμούς DRR |

**Περιγραφή Αποτελεσμάτων για τις Memory Accesses**

Αυτό που γίνεται εμφανές από τα παραπάνω αποτελέσματα είναι ότι η χρήση dynamic array στο package list έχει ως αποτέλεσμα περισσότερες προσβάσεις στην μνήμη συγκρινόμενη πάντα με τις άλλες δύο δομές δεδομένων (Single-Linked-List και Double-Linked-List). Οι υπόλοιποι συνδυασμοί οδηγούν σε παρόμοια αποτελέσματα με καλύτερη από όλες αυτή με χρήση Single-List και σε client list και σε package list.

**Περιγραφή Αποτελεσμάτων για τις Memory Footprint**

Αναφορικά με το memory footprint παρατηρούμε μεγάλη αύξηση σε χρήση dynamic-array, λιγότερη χρήση στη double-linked-list και ακόμα λιγότερη στην single-linked-list. Αυτό γίνεται εξαιτίας του δυναμικού allocation και deallocation κόμβων στις δομές δεδομένων. Για αυτό το λόγο και ο dynamic-array κάνει μεγαλύτερο allocation μνήμης εξαιτίας του συνεχόμενου resizing που πραγματοποιεί στην υλοποίηση του.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | Τύπος | Συνδυασμός | Τιμή | | Access | SLL -> SLL | 74843453 | | Footprint | DA -> SLL | 821.3 KB | |
| Πίνακας 2. Βέλτιστος συνδυασμός |

Όπως παρατηρούμε, ο βέλτιστος συνδυασμός υλοποιήσεων που έχει τις λιγότερες προσβάσεις στην μνήμη είναι με τη χρήση απλής συνδεδεμένης λίστας (SLL) τόσο για τους κόμβους όσο και για τα πακέτα.

Για το αποτύπωμα στην μνήμη η καλύτερη επιλογή́ είναι δυναμικός πίνακας (DA) για τους κόμβους και απλή́ συνδεδεμένη λίστα (SLL) για τα πακέτα.

# Dijkstra

Αρχικά, τρέχουμε τον κώδικα και τον Dijkstra χωρις την υλοποίηση της DDTR. Τα αποτελέσματα που παράγονται είναι :

Shortest path is 1 in cost. Path is: 0 41 45 51 50

Shortest path is 0 in cost. Path is: 1 58 57 20 40 17 65 73 36 46 10 38 41 45 51

Shortest path is 1 in cost. Path is: 2 71 47 79 23 77 1 58 57 20 40 17 52

Shortest path is 2 in cost. Path is: 3 53

Shortest path is 1 in cost. Path is: 4 85 83 58 33 13 19 79 23 77 1 54

Shortest path is 3 in cost. Path is: 5 26 23 77 1 58 99 3 21 70 55

Shortest path is 3 in cost. Path is: 6 42 80 77 1 58 99 3 21 70 55 56

Shortest path is 0 in cost. Path is: 7 17 65 73 36 46 10 58 57

Shortest path is 0 in cost. Path is: 8 37 63 72 46 10 58

Shortest path is 1 in cost. Path is: 9 33 13 19 79 23 77 1 59

Shortest path is 0 in cost. Path is: 10 60

Shortest path is 5 in cost. Path is: 11 22 20 40 17 65 73 36 46 10 29 61

Shortest path is 0 in cost. Path is: 12 37 63 72 46 10 58 99 3 21 70 62

Shortest path is 0 in cost. Path is: 13 19 79 23 77 1 58 99 3 21 70 55 12 37 63

Shortest path is 1 in cost. Path is: 14 38 41 45 51 68 2 71 47 79 23 77 1 58 33 13 92 64

Shortest path is 1 in cost. Path is: 15 13 92 94 11 22 20 40 17 65

Shortest path is 3 in cost. Path is: 16 41 45 51 68 2 71 47 79 23 77 1 58 33 32 66

Shortest path is 0 in cost. Path is: 17 65 73 36 46 10 58 33 13 19 79 23 91 67

Shortest path is 1 in cost. Path is: 18 15 41 45 51 68

Shortest path is 2 in cost. Path is: 19 69

Εισάγουμε την βιβλιοθήκη DDTR και τρέχουμε όλες τις διαθέσιμες δομές δεδομένων (SLL, DLL, και DA αντίστοιχα), εκτελώντας στην συνέχεια το εργαλείο Valgrind βρίσκοντας τις προσβάσεις μνήμης και το αποτύπωμα μνήμης.

**Περιγραφή Αποτελεσμάτων**

Στον παρακάτω πίνακα μπορούμε να παρατηρήσουμε τα αποτελέσματα των μετρήσεων για τη πρόσβαση στη μνήμη και το αποτύπωμα μνήμης για τις 3 διαφορετικές υλοποίησης του αλγορίθμου. Παρατηρούμε ότι για την πρόσβαση στην μνήμη η αλληλουχία είναι : SLL 🡪 DLL 🡪 DA ενώ για το αποτύπωμα μνήμης είναι : SLL 🡪 DA 🡪 DLL.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | Δομή Δεδομένων | Mem-Access | Mem-Footprint | | Απλά Συνδεδεμένη Λίστα (SLL) | 113274899 | 467.5 KB | | Διπλά Συνδεδεμένη Λίστα (DLL) | 113449694 | 582.1 KB | | Δυναμικός Πίνακας (DA) | 152153391 | 471.7 KB | |
| Πίνακας 3. Dijkstra αποτελέσματα |

|  |
| --- |
| Chart, scatter chart  Description automatically generated |
| Διάγραμμα 2. Mem-Accesses και Mem-Footprint για τους συνδυασμούς Dijkstra |

Όπως παρατηρούμε, η καλύτερη δομή δεδομένων τόσο στον αριθμό των προσβάσεων στην μνήμη (Memory-Accesses) όσο και αποτύπωμα μνήμης είναι η απλή συνδεδεμένη λίστα (SLL).