ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδιασμός Ενσωματωμένων Συστημάτων
9ο Εξαμηνο ΗΜΜΥ

Συνεργάτες: Γεώργιος-Ταξιάρχης Γιαννιός, Α.Μ.: 03116156
Μπέτζελος Χρήστος, Α.Μ.: 03116067

2η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

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

Άσκηση 1: Βελτιστοποίηση δυναμικών δομών δεδομένων του αλγορίθμου DRR

Ο source code του DRR έχει ήδη περασμένη την βιβλιοθήκη DDTR.

a) Αρχικά εκτελέσαμε την εφαρμογή με όλους τους διαφορετικούς συνδυασμούς υλοποιήσεων δομών δεδομένων για τη λίστα των πακέτων και τη λίστα των κόμβων. Αυτό έγινε εφικτό τοποθετώντας σε σχόλια κάθε φορά τα define που αφορουν δυο δομές δεδομένων, και αφήνοντας εκτός τη δομή που μελετούσαμε. Παρακάτω φαίνονται οι εντολές μέσω των οποίων κάναμε Compilation της εφαρμογής DRR, για 9 διαφορετικούς συνδυασμούς.

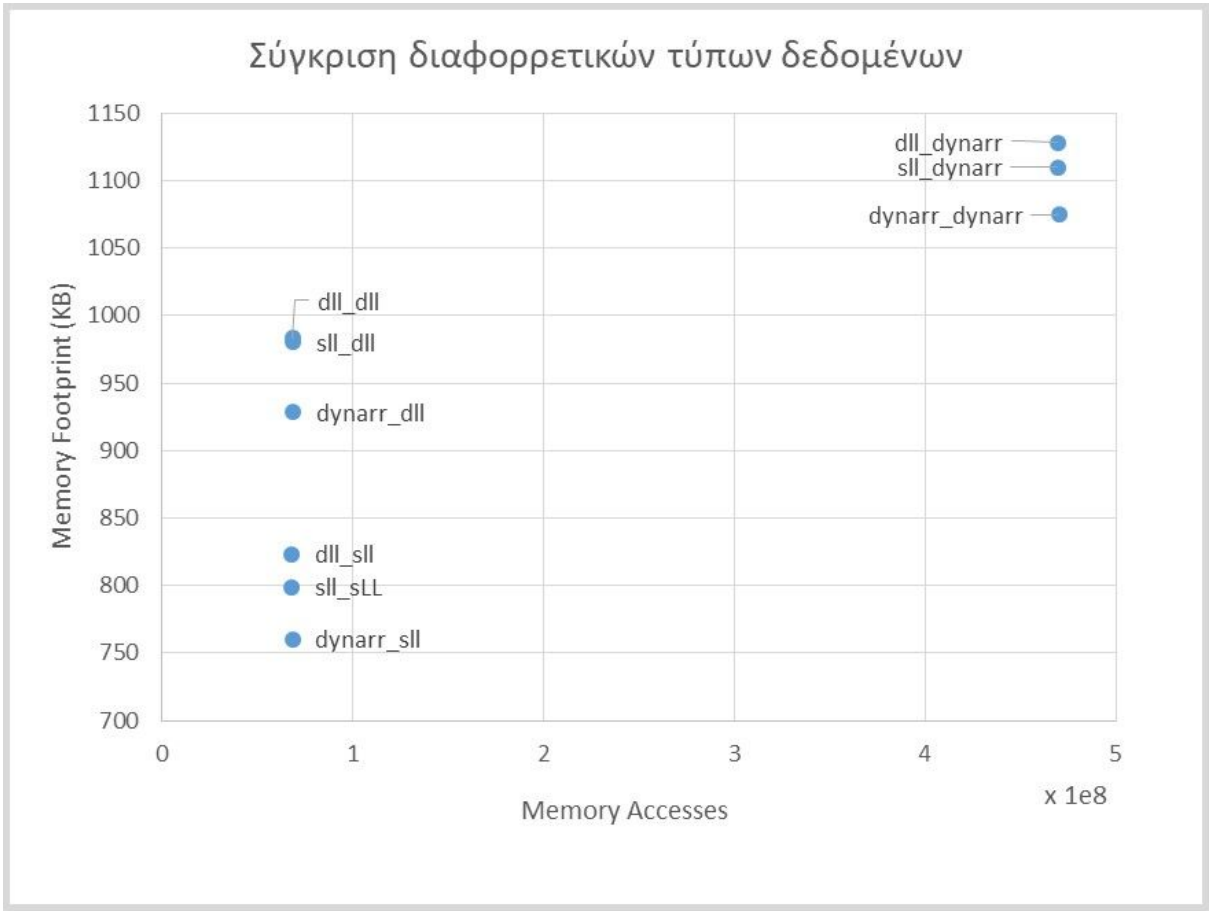
```
gcc-4.8 drr.c -o drr_sll_sll -pthread -lcdsl -L./../synch_implementations
-I./../synch_implementations

gcc-4.8 drr.c -o drr_sll_dynarr -pthread -lcdsl -L./../synch_implementations
-I./../synch_implementations

gcc-4.8 drr.c -o drr_dll_sll -pthread -lcdsl -L./../synch_implementations
-I./../synch_implementations

gcc-4.8 drr.c -o drr_dll_dll -pthread -lcdsl -L./../synch_implementations
-I./../synch_implementations
```

```
gcc-4.8 drr.c -o drr_dll_dynarr -pthread -lcdsl -L./../synch_implementations
-I./../synch_implementations

gcc-4.8 drr.c -o drr_dynarr_sll -pthread -lcdsl -L./../synch_implementations
-I./../synch_implementations

gcc-4.8 drr.c -o drr_dynarr_dll -pthread -lcdsl -L./../synch_implementations
-I./../synch_implementations

gcc-4.8 drr.c -o drr_dynarr_dynarr -pthread -lcdsl -L./../synch_implementations
-I./../synch_implementations
```

Χρησιμοποιήσαμε το *script.sh* και για κάθε συνδυασμό καταγράψαμε τα αποτελέσματά του σχετικά με τον αριθμό των προσβάσεων στη μνήμη (memory accesses) και το μέγεθος της απαιτούμενης μνήμης (memory footprint).

Node Struct	Packet Struct	Memory Accesses	Memory Footprint
SLL	SLL	67698178	798.8 KB
SLL	DLL	68335279	980.3 KB
SLL	DYN_ARR	469412656	1.110 MB
DLL	SLL	67710604	823.3 KB
DLL	DLL	68347313	983.3 KB
DLL	DYN_ARR	469428855	1.128 MB
DYN_ARR	SLL	68225743	760.2 KB
DYN_ARR	DLL	68884212	928.5 KB
DYN_ARR	DYN_ARR	470133885	1.075 MB

Παρουσιάζουμε τα αποτελέσματα και σε ένα scatter plot διάγραμμα.



b) Ο συνδυασμός υλοποιήσεων δομών δεδομένων με τον οποίο η εφαρμογή έχει τον μικρότερο αριθμό προσβάσεων στη μνήμη (**minimum number of memory accesses**) είναι ο SLL-SLL.

c) Ο συνδυασμός υλοποιήσεων δομών δεδομένων με τον οποίο η εφαρμογή έχει τις μικρότερες απαιτήσεις σε μνήμη (**smaller memory footprint**) είναι ο DYNARR-SLL.

Άσκηση 2: Βελτιστοποίηση δυναμικών δομών δεδομένων του αλγορίθμου Dijkstra

Ο αλγόριθμος dijkstra βρίσκει τη συντομότερη διαδρομή (shortest path) σε έναν πίνακα μεγέθους 100x100. Οι κόμβοι που εξετάζει και αποτελούν τη συντομότερη διαδρομή (shortest path) αποθηκεύονται σε μια λίστα. Εφαρμόζουμε τη μεθοδολογία DDTR για την εφαρμογή αυτή:

a) Εκτελέσαμε την εφαρμογή και καταγράψαμε τα αποτελέσματα:

```
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
```

b) Αφού κάναμε εισαγωγή της βιβλιοθήκης στην εφαρμογή, αντικαταστήσαμε τη δομή δεδομένων της, με τις δομές δεδομένων της βιβλιοθήκης. Πιο ειδικά οι αλλαγές/προσθήκες που πραγματοποιήθηκαν ήταν οι εξής:

i) Προσθήκη στην αρχή των απαραίτητων define της βιβλιοθήκης DDTR

```
#include <stdlib.h>
//#define SLL
//#define DLL
#define DYN_ARR

#if defined(SLL)
#include "../synch_implementations/cdsl_queue.h"
#endif
#if defined(DLL)
#include "../synch_implementations/cdsl_deque.h"
#endif
#if defined(DYN_ARR)
#include "../synch_implementations/cdsl_dyn_array.h"
#endif
```

ii)Αντικαταστήσαμε την εντολή:

```
QITEM *qHead = NULL;
```

με τις εντολές:

```
#if defined(SLL)
cdsl_sll *qHead;
#endif
#if defined(DLL)
cdsl_dll *qHead;
#endif
#if defined(DYN_ARR)
cdsl_dyn_array *qHead;
#endif
```

ώστε ανάλογα με τη δομή δεδομένων που χρησιμοποιούμε, να ορίζεται ο κατάλληλος δείκτης σε αυτή.

iii)Στη main προσθέσαμε τις ακόλουθες εντολες:

```
#if defined(SLL)
      qHead = cdsl_sll_init();
#endif
#if defined(DLL)
      qHead = cdsl_dll_init();
#endif
#if defined(DYN_ARR)
      qHead = cdsl_dyn_array_init();
#endif
```

ώστε να αρχικοποιηθεί η κατάλληλη δομή δεδομένων με βάση τα define

iv)Μέρος της συνάρτησης **enque**, η οποία δημιουργούσε ένα κόμβο με βάση τις τιμές των ορισμάτων και τον προσέθετε στο τέλος, φαίνεται παρακάτω:

```
QITEM *qLast = qHead;
qNew->qNext = NULL;

  if (!qLast)
      {
      qHead = qNew;
      }
  else
      {
      while (qLast->qNext) qLast = qLast->qNext;
      qLast->qNext = qNew;
      }
```

Αντικαταστήσαμε τις παραπάνω εντολές,με την εντολή της βιβλιοθήκης:

```
qHead->enqueue(0, qHead, (void *)qNew);
```

,η οποία κάνει την ίδια εισαγωγή κόμβου, ανεξάρτητα απο τη δομή δεδομένων που χρησιμοποιούμε.

v)Στη συνάρτηση **deque** αντικαταστήσαμε την παρακάτω εντολή, η οποία δημιουργούσε ένα pointer, ο οποίος έδειχνε στο ίδιο σημείο (ίδια διεύθυνση μνήμης) που δείχνει ο qHead,

```
QITEM *qKill = qHead;
```

με τις εντολές:

```
it = qHead->iter_begin(qHead);
QITEM *qKill = (QITEM *)qHead->iter_deref(qHead, it);
```

που επιτελουν την ίδια λειτουργία ανεξάρτητα απο τη δομή δεδομένων (ορίζεται δείκτης qKill που δείχνει, όπου δείχνει το qHead)

Τέλος για να γίνει η διαγραφή του πρώτου κόμβου, χρειάστηκε να αντικατασταθουν οι παρακάτω εντολές:

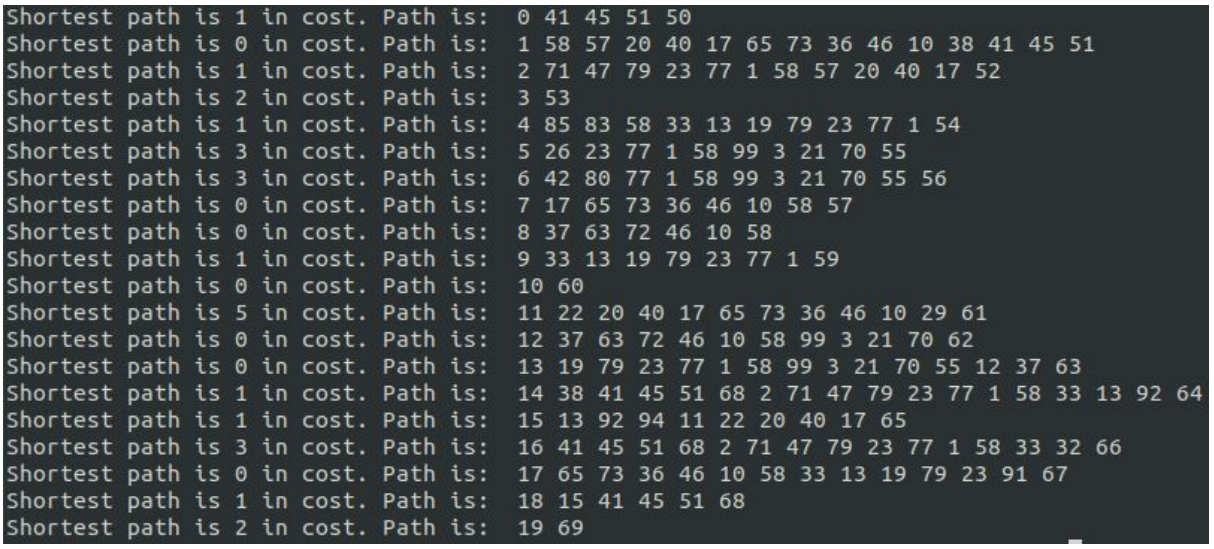
```
qHead = qHead->qNext;
free(qKill);
```

με την εντολή της βιβλιοθήκης:

```
qHead->remove(0, qHead, qKill);
```

Εκτελέσαμε την εφαρμογή και βεβαιωθήκαμε ότι η βιβλιοθήκη έχει εισαχθεί σωστά, συγκρίνοντας τα αποτελέσματα που παράγονται με αυτά που καταγράψατε στο ερώτημα (a). (Προφανώς είναι ακριβώς τα ίδια).

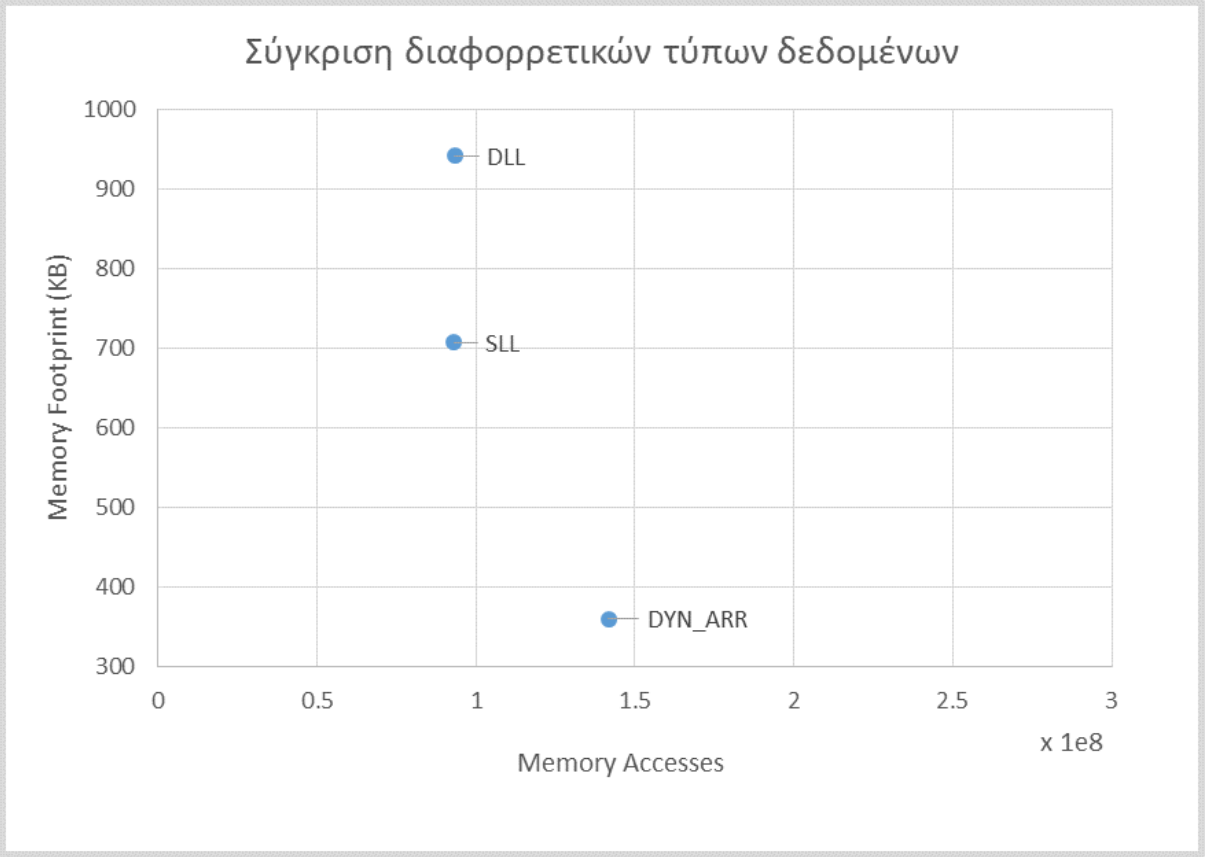
```
gcc-4.8    dijkstra_opt.c    -o    dijkstra_opt_sll    -pthread    -lcdsl
-L./../synch_implementations -I./../synch_implementations

gcc-4.8    dijkstra_opt.c    -o    dijkstra_opt_dll    -pthread    -lcdsl
-L./../synch_implementations -I./../synch_implementations

gcc-4.8    dijkstra_opt.c    -o    dijkstra_opt_dynarr    -pthread    -lcdsl
-L./../synch_implementations -I./../synch_implementations
```



c) Εκτελέσαμε με το αρχείο *script.sh* την εφαρμογή για τις εξής δυναμικές δομές δεδομένων: Απλά Συνδεδεμένη Λίστα – Single Linked List (SLL), Διπλά Συνδεδεμένη Λίστα – Double Linked List (DLL) και Δυναμικό Πίνακα – Dynamic Array (DYN_ARR). Καταγράψαμε τα αποτελέσματα του αριθμού προσβάσεων στη μνήμη (memory accesses) και του μέγιστου μεγέθους μνήμης που απαιτείται για κάθε υλοποίηση που δοκιμάσαμε (memory footprint).

Struct	Memory Accesses	Memory Footprint
SLL	92993000	707.7 KB
DLL	93199871	941.8 KB
DYN_ARR	141741903	360.7 KB

Παρουσιάζουμε τα αποτελέσματα και σε ένα scatter plot διάγραμμα.



d) Η υλοποιήση δομής δεδομένων με την οποία η εφαρμογή έχει τον μικρότερο αριθμό προσβάσεων στη μνήμη (**lowest number of memory accesses**) είναι η SLL.
e) Η υλοποιήση δομής δεδομένων με την οποία η εφαρμογή έχει μικρότερες απαιτήσεις σε μνήμη (**lowest memory footprint**) είναι η DYN_ARR.