ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΙΣΤΩΝ

ΣΧΕΔΙΑΣΜΟΣ ΕΝΣΩΜΑΤΩΜΕΝΩΝ ΣΥΣΤΗΜΑΤΩΝ – 7ο ΕΞΑΜΗΝΟ

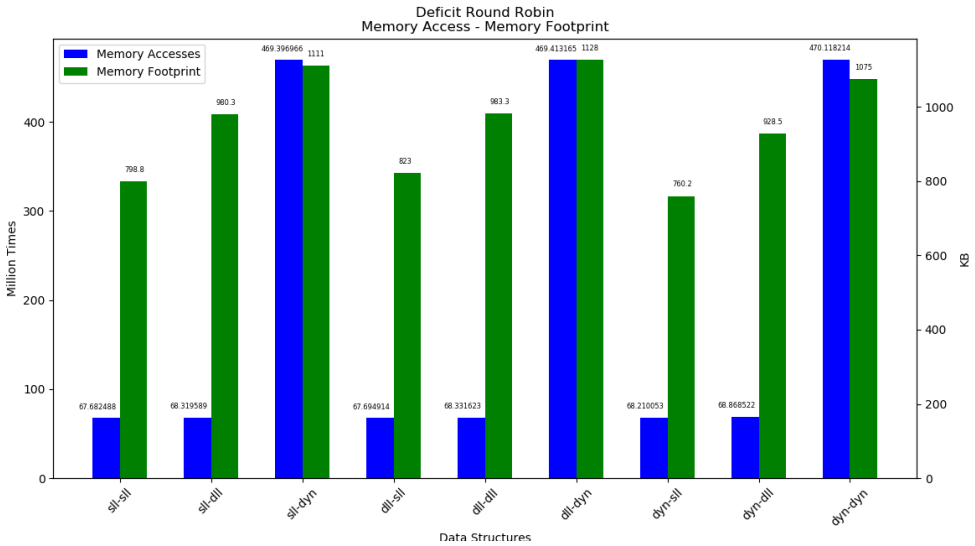
Μαρκέτος Νικόδημος – ΑΜ:03117095

Τζομάκα Αφροδίτη – ΑΜ: 03117107

2η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

ΑΣΚΗΣΗ 1

a. Εκτελώντας μέσω ενός script τις εντολές/εργαλεία του valgrind για τον δοθέντα κώδικα και για κάθε δυνατό συνδυασμό παίρνουμε τα αποτελέσματα που φαίνονται στο παρακάτω διάγραμμα. Το διάγραμμα περιέχει πληροφορίες τόσο για τα memory accesses όσο και για το memory footprint:



b. Βλέπουμε ότι την καλύτερη απόδοση λαμβάνοντας υπόψη τα memory accesses εμφανίζει ο συνδυασμός single linked list – single linked list με 67682488 προσβάσεις.

c. Εδώ, παίρνοντας ως μετρική το memory footprint, καλύτερη απόδοση φαίνεται να μας δίνει ο συνδυασμός dynamic array – single linked list με 760.2 KB.

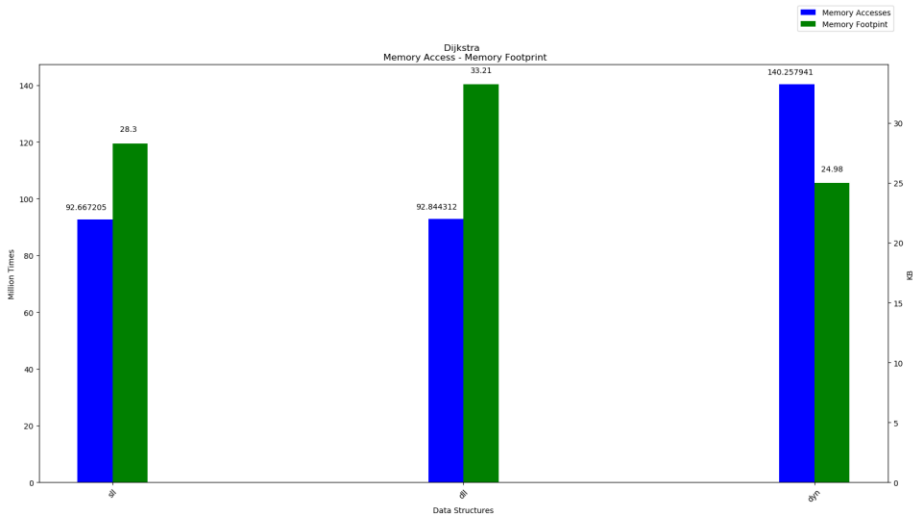
ΑΣΚΗΣΗ 2

a. Εκτελώντας τον αρχικό αλγόριθμο με τα δοθέντα δεδομένα παίρνουμε τα εξής αποτελέσματα:

```
aphrodite@aphrodite-Vostro-15-3568:~/Embedded/Lab2/DDTR/dijsktra$ ./d input.dat
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. Ο τροποποιημένος κώδικας που εμπεριέχει την βιβλιοθήκη DDTR παρατίθεται στο τέλος αυτής της αναφοράς. Ελέγχουμε την ορθότητά του και βλέπουμε ότι ξανά, δίνει τα σωστά αποτελέσματα (εκείνα που καταγράφηκαν και παραπάνω).

c. Εκτελώντας ξανά τον κώδικα και τις εντολές valgrind μέσω ενός script για κάθε δυνατό συνδυασμό παίρνουμε τα αποτελέσματα που φαίνονται στο παρακάτω διάγραμμα τόσο για τα memory accesses όσο και για το memory footprint:



d. Βλέπουμε ότι την καλύτερη απόδοση λαμβάνοντας υπόψη τα memory accesses εμφανίζει η υλοποίηση με single linked list με 92667205 προσβάσεις.

e. Τώρα παίρνοντας ως μετρική το memory footprint καλύτερη απόδοση φαίνεται να μας δίνει η υλοποίηση με dynamic array με 24.98 KB.

Κώδικας Άσκησης 2:

```
#include <stdio.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

#define NUM_NODES 100
#define NONE 9999

struct _NODE
{
  int iDist;
  int iPrev;
};
typedef struct _NODE NODE;

struct _QITEM
{
  int iNode;
  int iDist;
  int iPrev;
  struct _QITEM *qNext;
};
typedef struct _QITEM QITEM;

#if defined(SLL)
iterator_cdsl_sll it, end;
#elif defined(DLL)
iterator_cdsl_dll it, end;
#else
iterator_cdsl_dyn_array it, end;
#endif

#if defined(SLL)
cdsl_sll *qHead;
#elif defined(DLL)
cdsl_dll *qHead;
#else
cdsl_dyn_array *qHead;
#endif



int AdjMatrix[NUM_NODES][NUM_NODES];

int g_qCount = 0;
NODE rgnNodes[NUM_NODES];
int ch;
int iPrev, iNode;
int i, iCost, iDist;

void print_path(NODE *rgnNodes, int chNode)
{
  if (rgnNodes[chNode].iPrev != NONE)
  {
```

```
    print_path(rgnNodes, rgnNodes[chNode].iPrev);
  }
  printf(" %d", chNode);
  fflush(stdout);
}

void enqueue(int iNode, int iDist, int iPrev)
{
  QITEM *qNew = (QITEM *)malloc(sizeof(QITEM));

  if (!qNew)
  {
    fprintf(stderr, "Out of memory.\n");
    exit(1);
  }
  qNew->iNode = iNode;
  qNew->iDist = iDist;
  qNew->iPrev = iPrev;

  qHead->enqueue(0, qHead, (void *)qNew);
  g_qCount++;
}

void dequeue(int *piNode, int *piDist, int *piPrev)
{
  it = qHead->iter_begin(qHead);
  QITEM *temp = (QITEM *)(qHead->iter_deref(qHead, it));
  if (g_qCount != 0)
  {
    *piNode = temp->iNode;
    *piDist = temp->iDist;
    *piPrev = temp->iPrev;
    g_qCount--;
  }

  qHead->dequeue(0, qHead);

  free(temp);
 }

int qcount(void)
{
  return (g_qCount);
}

int dijkstra(int chStart, int chEnd)
{

  for (ch = 0; ch < NUM_NODES; ch++)
  {
    rgnNodes[ch].iDist = NONE;
    rgnNodes[ch].iPrev = NONE;
  }

  if (chStart == chEnd)
  {
    printf("Shortest path is 0 in cost. Just stay where you are.\n");
  }
  else
  {
    rgnNodes[chStart].iDist = 0;
    rgnNodes[chStart].iPrev = NONE;

    enqueue(chStart, 0, NONE);

    while (qcount() > 0)
    {
      dequeue(&iNode, &iDist, &iPrev);
      for (i = 0; i < NUM_NODES; i++)
      {
        if ((iCost = AdjMatrix[iNode][i]) != NONE)
        {
          if ((NONE == rgnNodes[i].iDist) ||
```

```
              (rgnNodes[i].iDist > (iCost + iDist)))
          {
            rgnNodes[i].iDist = iDist + iCost;
            rgnNodes[i].iPrev = iNode;
            enqueue(i, iDist + iCost, iNode);
          }
        }
      }
    }

    printf("Shortest path is %d in cost. ", rgnNodes[chEnd].iDist);
    printf("Path is: ");
    print_path(rgnNodes, chEnd);
    printf("\n");
  }
}

int main(int argc, char *argv[])
{
  int i, j, k;
  FILE *fp;

  #if defined(SLL)
  qHead =  cdsl_sll_init();
  #elif defined(DLL)
  qHead =  cdsl_dll_init();
  #else
  qHead =  cdsl_dyn_array_init();
  #endif

  if (argc < 2)
  {
    fprintf(stderr, "Usage: dijkstra <filename>\n");
    fprintf(stderr, "Only supports matrix size is #define'd.\n");
  }

  /* open the adjacency matrix file */
  fp = fopen(argv[1], "r");

  /* make a fully connected matrix */
  for (i = 0; i < NUM_NODES; i++)
  {
    for (j = 0; j < NUM_NODES; j++)
    {
      /* make it more sparce */
      fscanf(fp, "%d", &k);
      AdjMatrix[i][j] = k;
    }
  }

  /* finds 10 shortest paths between nodes */
  for (i = 0, j = NUM_NODES / 2; i < 20; i++, j++)
  {
    j = j % NUM_NODES;
    dijkstra(i, j);
  }
  exit(0);
}
```