ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδιασμός Ενσωματωμένων Συστήματων

Άσκηση 2η

Αναστάσιος Στέφανος Αναγνώστου
03119051

Σαββίνα Νάστου
03119146

6 Δεκεμβρίου 2023

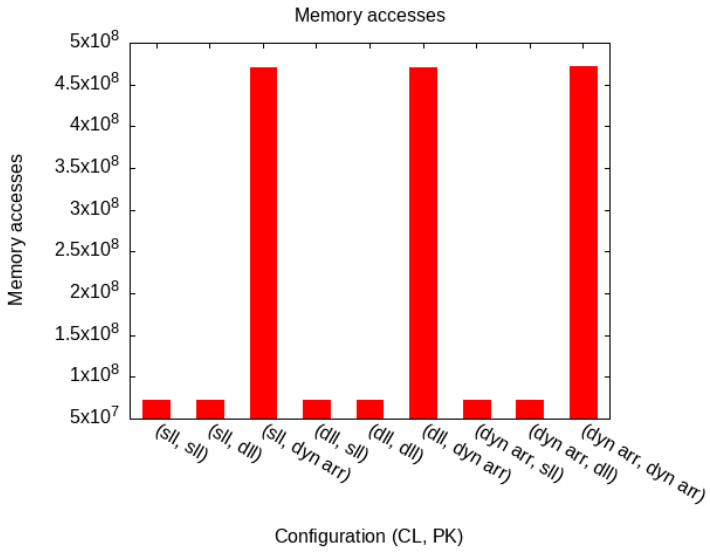
Περιεχόμενα

Ζητούμενο 1

1.1

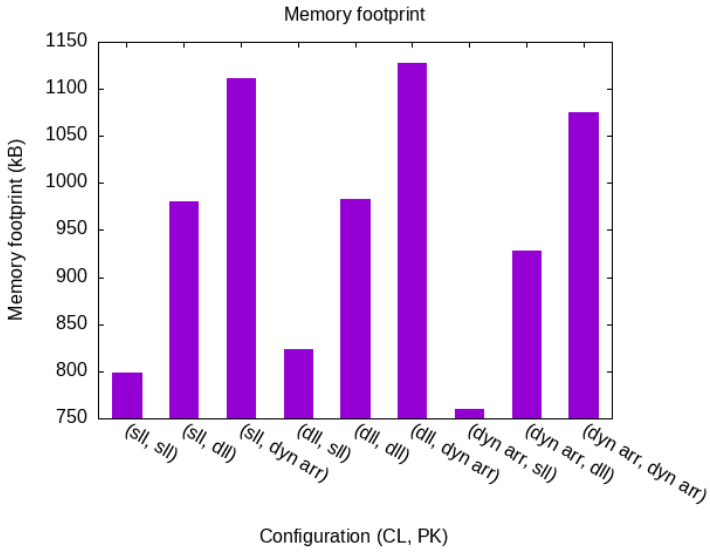
Η εκτέλεση της εφαρμογής για όλους τους διαφορετικούς συνδυασμοσύ υλοποιήσεων δομών δεδομένων και η αντίστοιχη καταγραφή των προσβάσεων στην μνήμη και της απαιτούμενης μνήμης έγινε χρήσει του script 1.

```
SLL_CL SLL_PK:        71911043
SLL_CL DLL_PK:        72583257
SLL_CL DYNARR_PK:     470964495
DLL_CL SLL_PK:        71923022
DLL_CL DLL_PK:        72595031
DLL_CL DYNARR_PK:     470979529
DYNARR_CL SLL_PK:     72409702
DYNARR_CL DLL_PK:     73094734
DYNARR_CL DYNARR_PK: 471712458
```



Σχήμα 1: Προσβάσεις στην μνήμη ανά συνδυασμό δομών

```
SLL_CL SLL_PK:         798.8
SLL_CL DLL_PK:         980.3
SLL_CL DYNARR_PK:      1111
DLL_CL SLL_PK:         823.0
DLL_CL DLL_PK:         983.3
DLL_CL DYNARR_PK:      1128
DYNARR_CL SLL_PK:      760.2
DYNARR_CL DLL_PK:      928.5
DYNARR_CL DYNARR_PK:   1075
```



Σχήμα 2: Απαίτηση μνήμης (KB) ανά συνδυασμό δομών

1.2

1.3

Ζητούμενο 2

2.1

Η έξοδος του προγράμματος 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
```

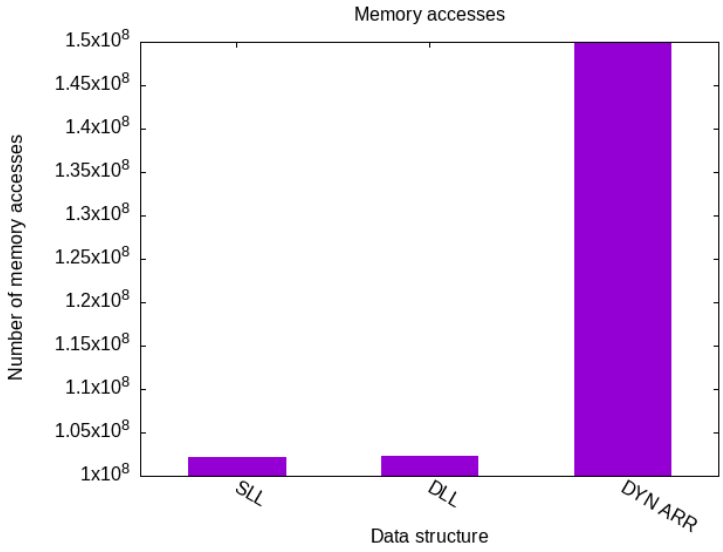
2.2

Οι αλλαγές στον κώδικα φαίνονται στο αρχείο 2.

2.3

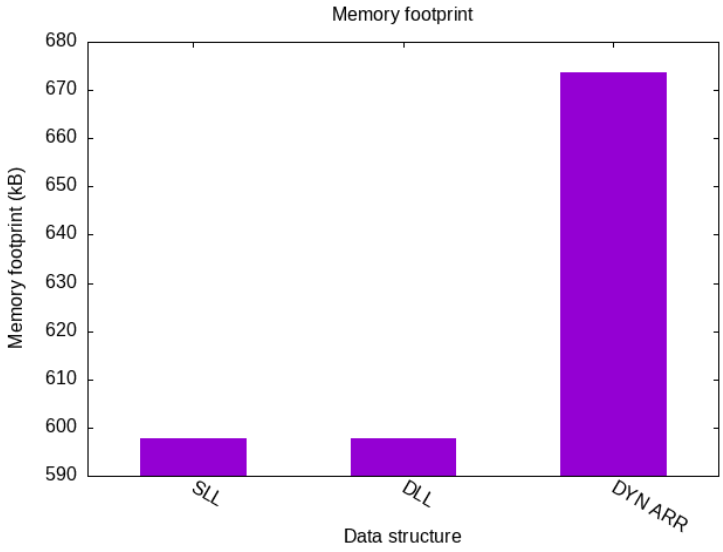
Οι εκτελέσεις των διάφορων παραλλαγών έγιναν χρήσει του script 3. Ανά εκτέλεση ελεγχόταν η έξοδος ώστε να επικυρώνεται η ορθότητα του προγράμματος.

```
SLL:     102183542
DLL:     102362751
DYNARR: 149983454
```



Σχήμα 3: Προσβάσεις στην μνήμη ανά δομή

```
SLL:     597.8
DLL:     597.8
DYNARR: 673.6
```



Σχήμα 4: Απαίτηση μνήμης (KB) ανά δομή

2.4

2.5

Παράρτημα 2

Listing 1: run-drr.sh

```
#!/bin/bash

link_dir="../synch_implementations"
include_dir="../synch_implementations"
gcc_options="-pthread -lcdsl -no-pie -L$link_dir -I$include_dir"

target="drr"
source="drr.c"
defs_cl=("SLL_CL" "DLL_CL" "DYN_ARR_CL")
defs_pk=("SLL_PK" "DLL_PK" "DYN_ARR_PK")

mem_log="mem_accesses_log.txt" # this gets overwritten in each iteration
accesses_out="mem_accesses.txt" # this stores the mem accesses of each config
mem_footprint="mem_footprint_log.txt" # this accumulates the mem logs
mem_footprints_out="mem_footprints.out" # this keeps the footprints from the logs


for def_cl in "${defs_cl[@]}"; do
    for def_pk in "${defs_pk[@]}"; do
        massif_out="massif_""$def_cl""_""$def_pk"".out"
        # compile with the current configuration
        gcc -D"$def_cl" -D"$def_pk" -o $target $source "$gcc_options" &&
        # get memory accesses
        valgrind --log-file="$mem_log" --tool=lackey --trace-mem=yes "./$target"
            &&
        grep -c 'I\| L' "$mem_log" >> "$accesses_out"
        # get memory footprint
        valgrind --tool=massif --massif-out-file="$massif_out" "./$target" &&
        ms_print "$massif_out" >> $mem_footprint
    done
done

grep '\^' $mem_footprint | cut -d'^' -f1 > $mem_footprints_out
```

Listing 2: dijkstra.c

```
#include <stdio.h>

#define NUM_NODES                           100
#define NONE                                9999

#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

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

struct _QITEM
{
    int iNode;
    int iDist;
    int iPrev;
#if defined(SLL)
        cdsl_sll *qNext;
#elif defined(DLL)
        cdsl_dll *qNext;
#else
        cdsl_dyn_array *qNext;
#endif
};
typedef struct _QITEM QITEM;

#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;

int qcount (void)
```

```
{
    return(g_qCount);
}

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));
    qNew->iNode = iNode;
    qNew->iDist = iDist;
    qNew->iPrev = iPrev;
    qNew->qNext = NULL;
    qHead->enqueue(0, qHead, (void*)qNew);
    g_qCount++;
}


void dequeue (int *piNode, int *piDist, int *piPrev)
{
    QITEM *temp = qHead->get_head(0, qHead);
    if (qHead)
    {
        *piNode = temp->iNode;
        *piDist = temp->iDist;
        *piPrev = temp->iPrev;
        qHead->dequeue(0, qHead);
        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;
        #if defined (SLL)
```

```
        qHead = cdsl_sll_init();
        #elif defined (DLL)
        qHead = cdsl_dll_init();
        #else
        qHead = cdsl_dyn_array_init();
        #endif
        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 (argc<2) {
        fprintf(stderr, "Usage: dijkstra <filename>\n");
        fprintf(stderr, "Only supports matrix size is #define'd.\n");
        return 0;
    }
    /* 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);
}
```

Listing 3: run-dijkstra.sh

```
#!/bin/bash

link_dir="../synch_implementations"
include_dir="../synch_implementations"
gcc_options="-pthread -lcdsl -no-pie -L$link_dir -I$include_dir"

source="dijkstra.c"
input="input.dat"
defs=("SLL" "DLL" "DYN_ARR")

footprints="mem_footprints.out"
accesses="mem_accesses.out"

correct_output="paths.out"

for def in "${defs[@]}"; do
    target="dijkstra_${def}"
    current_output="paths_${def}.out"

    # compile with the current configuration
    gcc -D"${def}" -o "${target}" "${source}" ${gcc_options} &&

    # run the program
    ./"${target}" "${input}" > "${current_output}"

    # compare the output with the correct one
    if diff -q "${correct_output}" "${current_output}" >/dev/null; then
        echo "The outputs are identical"
    else
        echo "The outputs are different"
    fi

    # get memory accesses
    temp_file=$(mktemp)
    valgrind --log-file="$temp_file" --tool=lackey --trace-mem=yes ./"${target}" "
        ${input}" > /dev/null
    grep -c 'I\| L' "$temp_file" >> "${accesses}"

    # get memory footprint
    valgrind --tool=massif --massif-out-file="${temp_file}" ./"${target}" "${input
        }" > /dev/null
    temp_file2=$(mktemp)
    ms_print "${temp_file}" > "${temp_file2}"
    grep '\^' "${temp_file2}" | cut -d'^' -f1 >> "${footprints}"
    rm "${temp_file}" "${temp_file2}"
done
```