Ministerul Educaţiei, Culturii și Cercetării

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Departamentul Ingineria Software și Automatică

RAPORT

La structuri de date și algoritmi

Lucrarea de laborator nr. 5

Varianta 18

A efectuat:

st. gr. TI-206 Cătălin Pleșu

A verificat:

Lector universitar Vitalie Mititelu

Chișinău 2021

Tema:
Algoritmi de prelucrare a listelor liniare simplu înlănțuite, listelor dublu înlănțuite, listelor circulare, cozilor și stivelor.

Scopul:
Obținerea deprinderilor practice de implementare și de utilizare a tipurilor abstracte de date „Listă liniară simplu înlănțuită”, „Listă liniară dublu înlănțuită”, „Listă circulară”, „Stivă”, „Coadă” în limbajul C cu asigurarea operațiilor de prelucrare de bază ale listei.

Sarcina:
Scrieți un program, care va tipări în ordine inversă subconsecutivitatea de numere dintre valoarea minimă și maximă ale unei liste simplu înlănțuită și se cere de a le diviza în 5 subliste.

Rezumat la temă:

Pentru a îndeplini această sarcină este necesar să îndeplinesc următorii pași:

1. Să utilizez lista simplu înlănțuită.
2. Să am în memorie o listă simplu înlănțuită fie că aceasta este:
    - citită de la tastatură
    - generată aleatoriu
3. Să identific elementul minim și maxim din această listă.
4. Să determin subconsecutivitatea ce se conține între aceste extreme.
5. Să inversez această listă selectată.
6. Să divizez lista inversată in 5 liste.

Implementarea acestor pași:

1. Am creat structura **list** cu doua variabile : **number** și **next**; variabila **next** este ceea ce face ca această structură să poată fi utilizată ca listă simplu înlănțuită.
2. Utilizatorul este înrebat daca dorește să înscrie de la tastatură o listă sau să i se genereze o listă aleatoare. Apoi este întrebat câte elemente dorește să conțină lista. Nu am considerat că este necesar să creez o funcție de citire de aceea lista este citită cu ajutorul unui for în caz ză utilizatorul dorește să introducă de la tastatură elementele listei utilizez funcția **get_num(int n)** care citește un număr. În caz că utilizatorul dorește o listă aleatoare folosesc funcția **random_range(int min, int max)** care returnează un număr random in intervalul **min** – **max**.
3. Pentru a determina numerele minime și maxime utilizez funcția **find_limits(list * head, list ** min, list ** max)** care primește capul listei si adresa la adresa elementeleor minime si maxime din main care la moment sunt nule. Funcția data selectează primul număr maxim sau minim și ultimul maxim sau minim în dependență de care a fost primul.
4. Pentru a nu afecta lista initiala creez o listă nouă pentru subconsecutivitatea din interval cu funcția **get_inner_list(list * head, list * min, list * max)**.
5. Inversez lista obținută cu funcția **reverse_list(list ** head)** care are ca argument adresa la adresa primului element.
6. Din lista inversata selectez un număr cat mai omogen de elemente și le pun în câte o listă cu ajutorul funcției **divide_in_n_lists(list * head, int n)** care are posibilitatea de a diviza o listă in oricare număr de liste.

Cod sursă:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

typedef struct node
{
int number;
struct node *next;
} list;

#define RED(string) "\x1b[31m" string "\x1b[0m"
#define GREEN(string) "\x1b[32m" string "\x1b[0m"

list *reverse_list(list **head);
list *forge_node(int number);
list **divide_in_n_lists(list *head, int n);
list *get_inner_list(list *head, list *min, list *max);

int get_num(int n);
int count_nodes(list *head);
int random_range(int min, int max);

void pirnt_list(list *head);
void print_n_lists(list **lists, int n);
void find_limits(list *head, list **min, list **max);

int main(int argc, char const *argv[])
{
srand(time(NULL));
printf("__Program_la_SDA__\n");
printf("1. Cititi o lista de la tastatura\n");
printf("2. Generati o lista aleatoare\n");
int option;
scanf("%d", &option);
printf("dati numarul de elemente ale listei\n");
int range;
scanf("%d", &range);
// citirea
int node;
int z = 1;
int abs = 500;
if (option == 1)
node = get_num(z);
if (option == 2)
node = random_range(-abs, abs);
```

```
list *head = forge_node(node);
list *t = head;
for (int i = 1; i < range; i++)
{
z++;
if (option == 1)
node = get_num(z);
if (option == 2)
node = random_range(-abs, abs);
t->next = forge_node(node);
t = t->next;
}
//determinarea min si max
list *max = NULL;
list *min = NULL;
find_limits(head, &min, &max);
// afisarea
printf("Lista citita:\n");
if (!head)
{
printf("lista este vida\n");
return 0;
}
list *temp = head;
while (temp)
{
if (temp == min || temp == max)
printf(GREEN("%d "), temp->number);
else if (temp->number == min->number || temp->number == max->number)
printf(RED("%d "), temp->number);
else
printf("%d ", temp->number);
temp = temp->next;
}
printf("\n");

printf("contine : %d elemente\n", count_nodes(head));
list *sub_list = get_inner_list(head, min, max);
reverse_list(&sub_list);
printf("MIN: %d\n", min->number);
printf("MAX: %d\n", max->number);
printf("subconsecutivitatea inversa cuprinsa intre %d si %d:\n ", min->number, max->number);
pirnt_list(sub_list);
node = count_nodes(sub_list);
printf("contine : %d elemente\n", node);
```

```
int divide = 5;
if (divide > node)
{
printf("nu se poate de divizat aceasta lista in %d liste fiind ca, contine doar %d\n", divide,
node);
return 0;
}
else
printf("\ncele %d liste obtinute: \n\n", divide);
list **lists;
lists = divide_in_n_lists(sub_list, divide);
print_n_lists(lists, divide);
return 0;
}

list *forge_node(int number)
{
list *node = (list *)malloc(sizeof(list));
node->number = number;
node->next = NULL;
return node;
}

int random_range(int min, int max)
{
int number = rand() % (max - min + 1) + min;
return number;
}

void pirnt_list(list *head)
{
if (!head)
{
printf("lista este vida\n");
return;
}
list *temp = head;
while (temp)
{
printf("%d ", temp->number);
temp = temp->next;
}
printf("\n");
}

void find_limits(list *head, list **min, list **max)
```

```
{
list *temp = head;
int first = 0;
(*min) = head;
(*max) = head;
temp = temp->next;
while (temp)
{
if (first == 0)
{
if (temp->number > (*max)->number)
{
(*max) = temp;
first = 1;
}
else if (temp->number < (*min)->number)
{
(*min) = temp;
first = -1;
}
}
if (first == 1)
{
if (temp->number > (*max)->number)
{
(*max) = temp;
first = -1;
}
if (temp->number <= (*min)->number)
{
(*min) = temp;
}
}
if (first == -1)
{
if (temp->number >= (*max)->number)
{
(*max) = temp;
}
if (temp->number < (*min)->number)
{
(*min) = temp;
first = 1;
}
}
```

```
temp = temp->next;
}
}

list *get_inner_list(list *head, list *min, list *max)
{
list *sub_head = NULL;
list *aux = NULL;
list *temp = head;
int inside = 0;
while (temp)
{
if (inside == 1)
if (sub_head == NULL)
{
sub_head = forge_node(temp->number);
aux = sub_head;
}
else
{
aux->next = forge_node(temp->number);
aux = aux->next;
}
if (temp == max || temp == min)
inside += 1;
temp = temp->next;
if ((temp == max || temp == min) && inside == 1)
break;
}
return sub_head;
}

list *reverse_list(list **head)
{
list *prev = NULL;
list *current = *head;
list *next = NULL;
while (current != NULL)
{
next = current->next;
current->next = prev;
prev = current;
current = next;
}
*head = prev;
}
```

```
int count_nodes(list *head)
{
list *temp = head;
int count = 0;
while (temp)
{
count += 1;
temp = temp->next;
}
return count;
}

list **divide_in_n_lists(list *head, int n)
{
list **aux = (list **)malloc(n * sizeof(list *));
list **liste = (list **)malloc(n * sizeof(list *));
for (int i = 0; i < n; i++)
liste[i] = NULL;
list *temp = head;
int nodes = count_nodes(head);
int count[n];
int remainder = nodes - (nodes / n) * n;
for (int i = 0; i < n; i++)
{
count[i] = nodes / n;
if (remainder > 0)
{
count[i] += 1;
remainder -= 1;
}
}
int cycle = 0;
int curent_list = 0;
while (temp)
{
if (cycle == count[curent_list])
{
curent_list++;
cycle = 0;
}
if (liste[curent_list] == NULL)
{
liste[curent_list] = forge_node(temp->number);
aux[curent_list] = liste[curent_list];
}
```

```
else
{
aux[curent_list]->next = forge_node(temp->number);
aux[curent_list] = aux[curent_list]->next;
}
temp = temp->next;
cycle++;
}

return liste;
}

int get_num(int n)
{
printf("dati elementul %d: ", n);
scanf("%d", &n);
return n;
}

void print_n_lists(list **lists, int n)
{
for (int i = 0; i < n; i++)
{
printf("lista %d: \n", i + 1);
pirnt_list(lists[i]);
printf("contine : %d elemente\n\n", count_nodes(lists[i]));
}
}
```

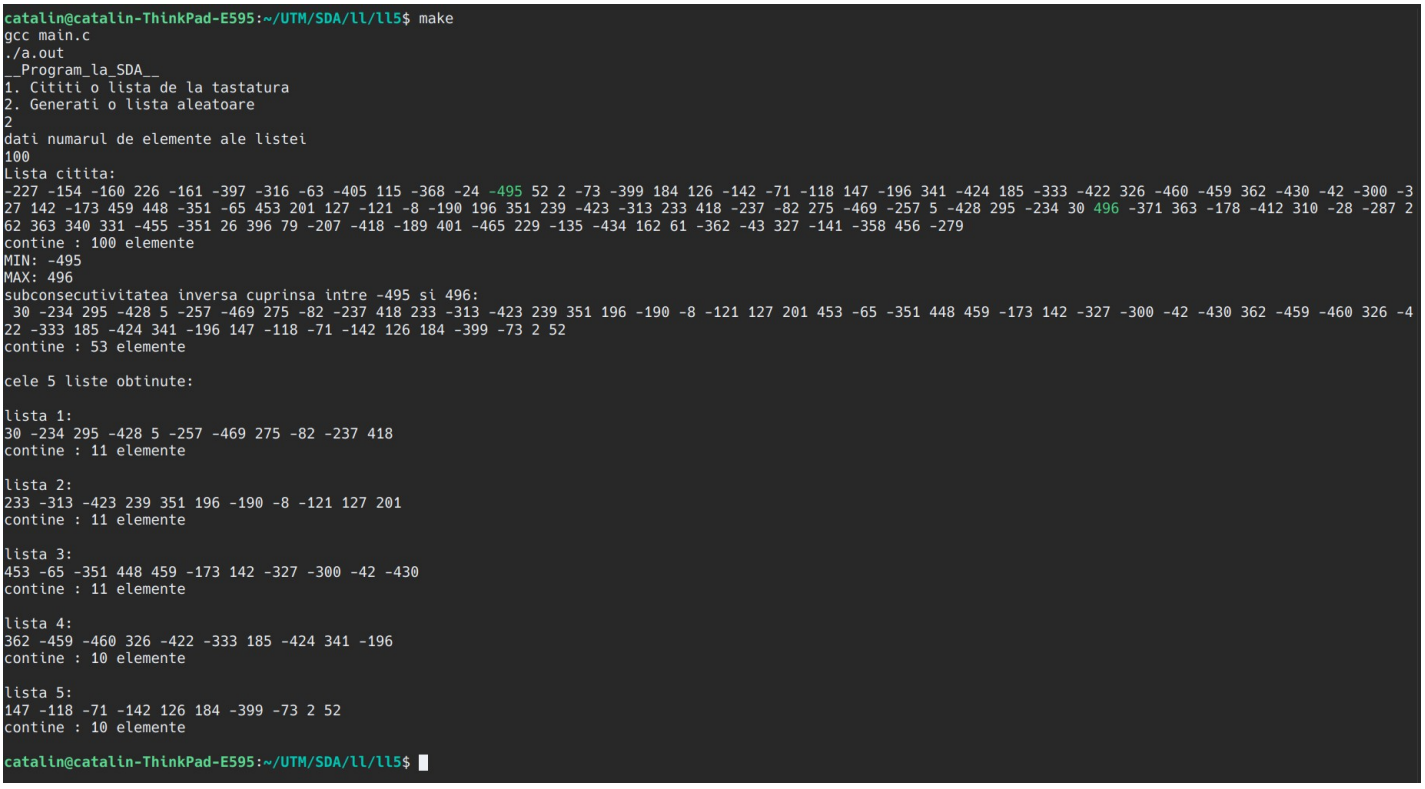
Testarea programului:

Acesta este outputul programului.
Primul pas a fost selectarea optiunii 2 apoi indicarea numărului de elemente 100.
Programul a generat această listă iar cu culoare verde sunt afișate elementele minim și maxim care au fost luate în considerare în caz ca avem alte elemente cu aceiați valoare dar nu sunt luate în considerare ele v-or fi afișate cu roșu.
Elementul minim și maxim în acest caz a fost -495 și 496.
Iar apoi submulțimea inversată pare să coincidă cu mulțimea de elemente din acest interval.
De notat submulțimea dată conține 53 de elemente număr care nu se împărte la 5 deci ultimele d liste au cu un element mai puțin decât primele 3.

```
catalin@catalin-ThinkPad-E595:~/UTM/SDA/ll/ll5$ make
gcc main.c
./a.out
__Program_la_SDA__
1. Cititi o lista de la tastatura
2. Generati o lista aleatoare
2
dati numarul de elemente ale listei
100
Lista citita:
-227 -154 -160 226 -161 -397 -316 -63 -405 115 -368 -24 -495 52 2 -73 -
399 184 126 -142 -71 -118 147 -196 341 -424 185 -333 -422 326 -460 -459
362 -430 -42 -300 -3 27 142 -173 459 448 -351 -65 453 201 127 -121 -8 -
190 196 351 239 -423 -313 233 418 -237 -82 275 -469 -257 5 -428 295 -234
30 496 -371 363 -178 -412 310 -28 -287 2 62 363 340 331 -455 -351 26 396
79 -207 -418 -189 401 -465 229 -135 -434 162 61 -362 -43 327 -141 -358
456 -279
contine : 100 elemente
MIN: -495
MAX: 496
subconsecutivitatea inversa cuprinsa intre -495 si 496:
30 -234 295 -428 5 -257 -469 275 -82 -237 418 233 -313 -423 239 351 196 -
190 -8 -121 127 201 453 -65 -351 448 459 -173 142 -327 -300 -42 -430 362
-459 -460 326 -4
22 -333 185 -424 341 -196 147 -118 -71 -142 126 184 -399 -73 2 52
contine : 53 elemente

cele 5 liste obtinute:

lista 1:
```

```
30 -234 295 -428 5 -257 -469 275 -82 -237 418
contine : 11 elemente

lista 2:
233 -313 -423 239 351 196 -190 -8 -121 127 201
contine : 11 elemente

lista 3:
453 -65 -351 448 459 -173 142 -327 -300 -42 -430
contine : 11 elemente

lista 4:
362 -459 -460 326 -422 -333 185 -424 341 -196
contine : 10 elemente

lista 5:
147 -118 -71 -142 126 184 -399 -73 2 52
contine : 10 elemente

catalin@catalin-ThinkPad-E595:~/UTM/SDA/ll/ll5$
```

Screenshot în caz ca textul nu este suficient.



```
1. Cititi o lista de la tastatura
2. Generati o lista aleatoare
1
dati numarul de elemente ale listei
10
dati elementul 1: 1
dati elementul 2: 2
dati elementul 3: 3
dati elementul 4: -5
dati elementul 5: 3
dati elementul 6: 9
dati elementul 7: 9
dati elementul 8: 9
dati elementul 9: -2
dati elementul 10: 9
Lista citita:
1 2 3 -5 3 9 9 9 -2 9
contine : 10 elemente
MIN: -5
MAX: 9
subconsecutivitatea inversa cuprinsa intre -5 si 9:
 -2 9 9 9 3
contine : 5 elemente

cele 5 liste obtinute:

lista 1:
-2
contine : 1 elemente

lista 2:
9
contine : 1 elemente

lista 3:
9
contine : 1 elemente

lista 4:
9
contine : 1 elemente

lista 5:
3
contine : 1 elemente
```

```
__Program_la_SDA__
1. Cititi o lista de la tastatura
2. Generati o lista aleatoare
2
dati numarul de elemente ale listei
200
Lista citita:
29 -40 44 -19 -25 42 4 -13 15 23 -40 48 11 21 32 -38 -39 -24 5 -3 30 -29 0 20 -3 -20 23 -44 -32 37 -50 47 48 44 44 39 35 -36 -25 -34 37 -15 -37 -3
23 46 -25 0 38 -4 13 18 17 -37 -13 30 -7 26 36 11 13 36 -26 -24 30 34 31 -20 48 -45 -38 0 -44 -24 48 -5 38 39 45 -9 -15 -26 -42 18 -13 -38 -36 -4 4
 50 -44 33 36 47 -42 -19 -3 39 -23 -6 10 -10 45 17 -18 8 -39 36 -38 23 -24 -37 47 -49 32 50 -37 12 -5 -17 11 -33 -34 -38 -36 41 -41 11 30 -47 21 -4
5 -41 -19 22 -9 5 0 43 18 39 -32 31 35 36 -22 0 15 40 45 49 1 -39 -36 -21 42 -46 -11 -32 0 -8 6 6 1 37 44 8 -8 -41 -34 -41 48 -50 6 -2 37 35 48 1 2
4 42 -1 41 20 -21 -30 11 -50 -25 -4 0 17 -49 22 34 38 -18 -9 46 -9
contine : 200 elemente
MIN: -50
MAX: 50
subconsecutivitatea inversa cuprinsa intre -50 si 50:
 32 -49 47 -37 -24 23 -38 36 -39 8 -18 17 45 -10 10 -6 -23 39 -3 -19 -42 47 36 33 -44 50 4 -4 -36 -38 -13 18 -42 -26 -15 -9 45 39 38 -5 48 -24 -44
0 -38 -45 48 -20 31 34 30 -24 -26 36 13 11 36 26 -7 30 -13 -37 17 18 13 -4 38 0 -25 46 23 -3 -37 -15 37 -34 -25 -36 35 39 44 44 48 47
contine : 84 elemente

cele 5 liste obtinute:

lista 1:
32 -49 47 -37 -24 23 -38 36 -39 8 -18 17 45 -10 10 -6 -23
contine : 17 elemente

lista 2:
39 -3 -19 -42 47 36 33 -44 50 4 -4 -36 -38 -13 18 -42 -26
contine : 17 elemente

lista 3:
-15 -9 45 39 38 -5 48 -24 -44 0 -38 -45 48 -20 31 34 30
contine : 17 elemente

lista 4:
-24 -26 36 13 11 36 26 -7 30 -13 -37 17 18 13 -4 38 0
contine : 17 elemente

lista 5:
-25 46 23 -3 -37 -15 37 -34 -25 -36 35 39 44 44 48 47
contine : 16 elemente
```

Concluzii:

1. Am însușit și mai bine operarea cu diverse TDA în special lista simplu înlănțuită.
2. Mi-am îmbunătățit deprinderile practice de a opera cu acest TDA.
3. Am elaborat diverse funcții cu scopuri diferite.
4. M-am distrat folosind funcția **rand**, pot afirma ca este una dintre placerile vieții.
5. Am observat că după efectuarea laboratoarelor mai complexe laboratorele care necesita programe de mai puțin de 300 de linii par foarte ușoare.
6. Am scris o mare parte a programului în funcția main fiind că nu am considerat necesar crearea unor funcții.
7. Nu consider că acest program ar trebui separat în mai multe fișiere.