Necula Madalina Andreea
412D

Lucrare 2 SDA

Pseudocod problema 1:

selection sort:

pentru i = first:last - 1

minim<-nodes[i]

gasesc minimul din subtabloul i:last, are indicele imin

interschimb nodes[i] si nodes[imin]

Cod +explicatii problema 1:

- Funcție de sortare prin selecție:

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

//functie de swap pentru doua int-uri
void swap(int *a, int *b)
{
   int aux = *a;
   *a = *b;
   *b = aux;
}
// pentru fiecare pozitie intre first si last cautam cel mai mic element din
// subtabloul ramas si il interschimbam cu pozitia curenta
void selectionSort(int nodes[], int first, int last, int (cmp)(int * a, int b))
{
   int pos_min, i, j, minim;
   for (i = first; i < last; i++) {
      pos_min = i;
```

```
        minim = nodes[i];
        for (j = i + 1; j < last; j++)
            if (cmp(&nodes[j], &minim) < 0) {
                minim = nodes[j];
                pos_min = j;
            }
        // interschimb nodes[i] cu nodes[pos_min]
        swap(&nodes[i], &nodes[pos_min]);
    }
}

int cmp_int(int *a, int *b) {
    return (*a) - (*b);
}

int main()
{
    // testez functia selectionSort
    int nodes[] = {0,4,3,7,9,1,3,0};
    selectionSort(nodes, 2, 7, cmp_int);
    int i;
    for (i = 2; i < 7; i++)
        printf("%d ", nodes[i]);
    return 0;
}
```

- Funcție de traversare in lățime a unui arbore binar:

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

```
// voi considera ca in fiecare nod al arborelui se afla un caracter
struct node_btree {
   char c;
   struct node_btree *left, *right; // copii nodului
};

// functie care creeza un nou nod din arborele binar
struct node_btree* newNode(char data)
{
   struct node_btree* node = (struct node_btree*)
                  malloc(sizeof(struct node_btree));
   node->c = data;
   node->left = NULL;
   node->right = NULL;

   return node;
}

void afisare_nivel(struct node_btree* r, int nivel);
int height(struct node_btree* node);
void afisare_pe_nivele(struct node_btree* r);

// functie care afiseaza arborele binar pe nivele
void afisare_pe_nivele(struct node_btree* r)
{
   int h = height(r);
   int i;
   for (i=1; i<=h; i++)
      afisare_nivel(r, i);
```

```
}

// functie care afiseaza un nivel din arbore
void afisare_nivel(struct node_btree* r, int nivel)
{
   if (r == NULL)
      return;
   if (nivel == 1)
      printf("%c", r->c);
   else if (nivel > 1)
   {
      afisare_nivel(r->left, nivel-1);
      afisare_nivel(r->right, nivel-1);
   }
}

// functie care determina inaltimea unui nod din arbore
int height(struct node_btree* node)
{
   if (node==NULL)
      return 0;
   else
   {
      // determin inaltimea fiecarui subarbore
      int lheight = height(node->left);
      int rheight = height(node->right);

      // returneaza inaltimea mai mare
      if (lheight > rheight)
```

```
        return(lheight+1);
      else return(rheight+1);
   }
}

void tl_tree(struct node_btree* r)
{
   afisare_pe_nivele(r);
}

int main()
{
   struct node_btree *r = newNode('a');
   r->left       = newNode('b');
   r->right      = newNode('c');
   r->left->left  = newNode('d');
   r->left->right = newNode('e');

   afisare_pe_nivele(r);

   return 0;
}
```

Cod+explicatii problema 2:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct element {
```

```
    int frecventa;
    char c;
};

// voi considera ca in fiecare nod al arborelui se afla un element de
// tip caracter-frecventa
struct node_btree {
    struct element elem;
    struct node_btree *left, *right; // copii nodului
};

// functie care creeza un nou nod din arborele binar
struct node_btree* newNode(struct element *data)
{
    struct node_btree* node = (struct node_btree*)
                    malloc(sizeof(struct node_btree));
    memcpy(node->elem, data, sizeof(struct element));
    node->left = NULL;
    node->right = NULL;

    return node;
}

void afisare_nivel(struct node_btree* r, int nivel);
int height(struct node_btree* node);
void afisare_pe_nivele(struct node_btree* r);

// functie care afiseaza arborele binar pe nivele
void afisare_pe_nivele(struct node_btree* r)
```

```
{
   int h = height(r);
   int i;
   for (i=1; i<=h; i++)
      afisare_nivel(r, i);
}

// functie care afiseaza un nivel din arbore
void afisare_nivel(struct node_btree* r, int nivel)
{
   if (r == NULL)
      return;
   if (nivel == 1)
      printf("%c", r->c);
   else if (nivel > 1)
   {
      afisare_nivel(r->left, nivel-1);
      afisare_nivel(r->right, nivel-1);
   }
}

// functie care determina inaltimea unui nod din arbore
int height(struct node_btree* node)
{
   if (node==NULL)
      return 0;
   else
   {
      // determin inaltimea fiecarui subarbore
```

```
        int lheight = height(node->left);
        int rheight = height(node->right);

        // returneaza inaltimea mai mare
        if (lheight > rheight)
            return(lheight+1);
        else return(rheight+1);
    }
}

void tl_tree(struct node_btree* r)
{
    afisare_pe_nivele(r);
}

//functie de swap pentru doua structuri de tip element
void swap(struct element *a, struct element *b)
{
    struct element aux = *a;
    *a = *b;
    *b = aux;
}

// pentru fiecare pozitie intre first si last cautam cel mai mic element din
// subtabloul ramas si il interschimbam cu pozitia curenta
void selectionSort(struct element *nodes, int first, int last,
                   int (*cmp)(int *a, int *b))
{
    int pos_min, i, j, minim;
```

```
    for (i = first; i < last; i++) {
        pos_min = i;
        minim = nodes[i]->frecventa;
        for (j = i + 1; j < last; j++)
            if (cmp(&nodes[j], &minim) < 0) {
                minim = nodes[j];
                pos_min = j;
            }
        // interschimb nodes[i] cu nodes[pos_min]
        swap(&nodes[i], &nodes[pos_min]);
    }
}

int cmp_int(int *a, int *b) {
    return (*a) - (*b);
}

int main()
{
    // citesc frecventele fiecarei litere din alfabet si construiesc alfabetul
    int i, frecv;
    struct element *alfabet = malloc (26 * sizeof(struct element));

    for (i = 0; i < 26; i++) {
        scanf("%d", &frecv);
        alfabet[i]->c = 'A' + i;
        alfabet[i]->frecventa = frecv;
    }
```

```
    // ordonez elementele din alfabet dupa frecventa
    selectionSort(alfabet, 0, 26, cmp_int);

    node_btree *root = newNode(alfabet[0]);

    // grupam elementele in arborele Huffman
    for (i = 0; i < 25; i++) {
        node_btree *node = newNode(alfabet[i]);
        node->left = alfabet[i];
        node->right = alfabet[i+1];
        node->elem.frecventa = alfabet[i]->frecventa + alfabet[i+1]->frecventa;
        node->elem.c = '\0';
    }
}
```