Subiecte lucrarea 1 SDA (09.04.2021)

Soluții propuse

1) Sa se scrie in limbaj C funcțiile ce implementează o coada (creare, adăugare, extragere, căutare, listare) ca lista dublu înlănțuită.

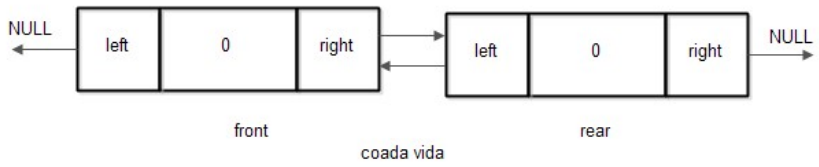
Se poate implementa coada dupa modelul cozii implementate ca lista simplu înlănțuite cu doua noduri false, front si rear. Inserarea se va face la stânga lui rear, iar extragerea se va face din dreapta lui front.

Principalele funcții si modul de apelare este prezentat in continuare. Explicațiile necesare nu au fost ilustrate, vezi model curs. Rezolvarea completa impune printscreen-uri din CLion cu testarea funcțiilor.

Codul scris cu roșu este preluat direct din exemplele date la curs.

Crearea cozii vide

```
struct dnode
{
    int info;
    struct dnode* left;
    struct dnode* right;
};

struct dnode* front;
struct dnode* rear;
```



Creare nod nou

```
struct dnode* new_dnode(int a)
{
    struct dnode* p;
    p=(struct dnode*)malloc(sizeof(struct dnode));
    p->info = a;
    p->left = NULL;
    p->right = NULL;
    return p;
}

// creare noduri false
front = new_dnode(0);
```

```
rear=new_dnode(0);
// legare noduri false (coada vida)
front->left=NULL;
front->right=rear;
rear->left=front;
rear->right=NULL;
```

Adăugare in coada

```
void add_l_dlist( struct dnode* p, int a)
{
    struct dnode* h;
    if (p!=NULL)
    {   h=new_dnode(a);
        h->left=p->left;
        h->right=p;
        p->left->right=h;
        p->left=h;
    }
}
```

Apelul funcției de inserare in coada:

```
add_l_dlist(rear,valoare);
```

Extragere din coada

```
int del_r_dlist( struct dnode* p)
{
    struct dnode* q;
    int x=-1;

    if (p!=NULL)
    {
        q=p->right;
        x = del_dlist(q);
    }
    return x;
}

int del_dlist( struct dnode* p)
{
    int x=-1;

    if (p!=NULL)
    {
        p->left->right=p->right;
        p->right->left=p->left;
        x= p->info;
        free(p);
    }
    return x;
}
```

Apelul funcției de extragere din coada:

```
del_r_dlist(front)
```

Căutare in coada

- *parametrii de intrare front, rear si valoarea nodului*
- *întoarce nodul găsit*

```
struct dnode* cauta(struct dnode* f, struct dnode* r, int a)
{
    struct dnode* p;
    struct dnode* res = NULL;

    p=f;
    while (p->right != r)
    {
        if (a == p->right->info) {
            res=p->right;
            break;
        }
        p=p->right;
    }
    return res;
}
```

Apelul funcției de căutare in coada:

```
cauta(front, rear, 1);
```

pentru inserarea de elemente neidentice:

```
if (cauta(front, rear, 1) == NULL) add_l_dlist(rear,1);
```

Listare coada

- *intrare front si rear*

```
void afisare(struct dnode* f, struct dnode* r)
{
    while (f->right != r)
    {
        printf("%d, ", f->right->info);
        f=f->right;
    }
    printf("\n");
}
```

Apelul functiei de listare

```
afisare(front,rear);
```

2) Sa se scrie in limbaj C funcțiile care creează, parcurge in ordine si caută dupa momentul de timp (calculat in intervalul de o zi) un nod pentru un arbore binar de căutare.

Un nod al arborelui este definit astfel:

```
struct bnode
{
int ora;
int min;
int sec;
struct bnode* left;
struct bnode* right;
}
```

Ordonarea arborelui se va face dupa momentul de timp (ora, minut, secunda).
Implementarea se va face dinamic.

Se poate folosi procedura de construire a unui arbore binar de căutare prezentata la curs. Procedura se va modifica pentru a face ordonarea dupa timp – se va calcula timpul in secunde apoi se va face căutarea si inserarea in arbore. Funcțiile de creare a unui nod nou si de parcurgere se vor adapta la noul tip de date al nodului.

Funcția de creare a unui nod nou in arborele binar

```
struct bnode* new_tree_node(int a)
{
    struct bnode* p;

    p= (struct bnode*) malloc(sizeof(struct bnode));
    p->data=a;
    p->left=NULL;
    p->right=NULL;
}
```

Funcția modificata de creare a unui nod nou in arborele binar

```
struct bnode* new_tree_node(int o, int m, int s)
{
    struct bnode* p;

    p= (struct bnode*) malloc(sizeof(struct bnode));
    p->ora=o;
    p->min=m;
    p->sec=s;
    p->left=NULL;
    p->right=NULL;
}
```

Funcția de creare a arborelui binar de căutare

```
struct bnode* build_abc(struct bnode*r, int a)
{
    if (r==NULL) r= new_tree_node(a);
    else
    {
        if (a < r->data ) r->left=build_abc(r->left,a);
        if (a > r->data ) r->right=build_abc(r->right,a);
    }
    return r;
}
```

Funcția modificata de creare a arborelui binar de căutare

```
struct bnode* build_abc(struct bnode*r, int o, int m, int s)
{
    int t1;
    int t2;
    if (r==NULL) r= new_tree_node(o,m,s);
    else
    {
        t1=(r->ora)*3600+(r->min)*60+r->sec;
        t2=o*3600+m*60+s;
        if (t2 < t1 ) r->left=build_abc(r->left,o, m,s);
        if (t2 > t1 ) r->right=build_abc(r->right,o,m,s);
    }
    return r;
}
```

Funcția de parcurgere in ordine a arborelui binar de căutare

```
void ldr(struct bnode* r)
{
    if(r!=NULL)
    {
        ldr(r->left);
        printf("%d, ", r->data);
        ldr(r->right);
    }
}
```

Funcția modificata de parcurgere in ordine a arborelui binar de căutare

```
void ldr(struct bnode* r)
{
    if(r!=NULL)
    {
        ldr(r->left);
        printf("%.2d : %.2d : %.2d\n", r->ora, r->min, r->sec);
        ldr(r->right);
    }
}
```

Funcția de căutare in arborele binar de căutare

```
struct bnode* search_abc(struct bnode*r, int a)
{
  if (r == NULL)  return NULL;
  if (r->data == a) return r;
  if (a < r->data) return (search_abc(r->left,a));
  if (a > r->data) return (search_abc(r->right,a));

}
```

Funcția modificata de căutare in arborele binar de căutare

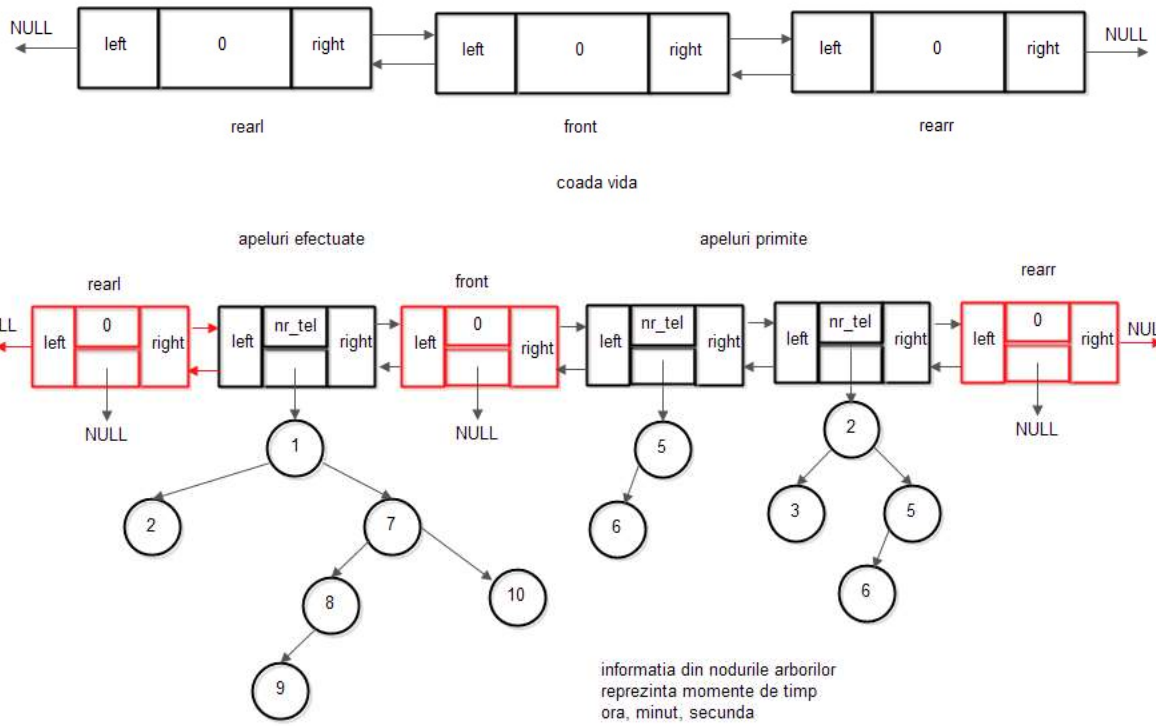
- *Intrare – rădăcina arborelui si informația din nod (ora, minut, secunda)*
- *Întoarce nodul găsit*

```
struct bnode* search_abc(struct bnode*r, int o, int m, int s)
{
    int t1;
    int t2;
    t1=(r->ora)*3600+(r->min)*60+r->sec;
    t2=o*3600+m*60+s;

    if (r == NULL)  return NULL;
    if (t1 == t2) return r;
    if (t2 < t1) return (search_abc(r->left,o, m, s));
    if (t2 > t1) return (search_abc(r->right,o, m, s));

}
```

1) Folosind obligatoriu funcțiile de la punctele 1) si 2) – eventual adaptate, sa se scrie funcțiile principale si modul lor de apelare pentru program de monitorizare (înregistrare si afișare) a apelurilor telefonice primite si efectuate. Se va utiliza o singura coada pentru ambele tipuri de apeluri. Se va înregistra numărul de telefon si apelurile primite sau efectuate in ordine cronologica. Numerele de telefon se vor înregistra o singura data pentru un tip de apel.

Ideea rezolvării:

Se poate crea o coada ca o lista dublu înlănțuita (vezi punctul), cu adăugare de noduri in stânga si in dreapta



- *apeluri primite - partea din dreapta a listei duble (intre front si rearr)*
- *adăugare in coada: inserare la stânga lui rearr*
- *extragere din coada: ștergere la dreapta lui front*
- *apeluri efectuate - partea din stânga a listei duble (intre front si rearl)*
- *adăugare in coada: inserare la dreapta lui rearl*
- *extragere din coada: ștergere la stânga lui front*
- *adăugările vor testa existenta nodului*
- *afișare apeluri: parcurgere ldr (ordine crescătoare) cu rădăcina pointerul citit din câmpul de info al nodului din coada*

Codul propus este:

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

/*
 * apeluri primite - partea din dreapta a listei duble (intre front si rearr)
 * - adaugare in coada -> inserare la stinga lui rearr, extragere din coada -
> stergere la dreapta lui front
 *
 * apeluri efectuate - partea din stinga a listei duble  (intre front si
rearl)
 * - adaugare in coada -> inserare la dreapta lui rearl, extragere din coada
-> stergere la stinga lui front
 *
 * adaugarile vor testa mai intii existenta nodului
 * cautare la stinga sau la dreapta lui front (intre front si rearr sau intre
forin si rearl)
 *
 * - afisare apeluri -> parcurgere ldr cu radacina pointerul citit din cimpul
de info al nodului din coada
 */

struct dnode
{
    struct bnode* info;
    int nr_tel;
    struct dnode* left;
    struct dnode* right;
};

struct dnode* front;
struct dnode* rearr;
struct dnode* rearl;


struct dnode* new_dnode(struct bnode* r, int a);
void add_r_dlist( struct dnode* p, struct bnode* r, int a);
void add_l_dlist( struct dnode* p, struct bnode* r, int a);
int del_r_dlist( struct dnode* p);
int del_l_dlist( struct dnode* p);
int del_dlist( struct dnode* p);
void afisarer(struct dnode* f, struct dnode* r);
void afisarel(struct dnode* f, struct dnode* r);
struct dnode* cautar(struct dnode* f, struct dnode* r, int a);
struct dnode* cautal(struct dnode* f, struct dnode* r, int a);

struct bnode {
    int ora;
    int min;
    int sec;
    struct bnode* left;
    struct bnode* right;
};

struct bnode* new_tree_node(int o, int m, int s);
```

```
struct bnode* build_abc(struct bnode*r, int o, int m, int s);
struct bnode* search_abc(struct bnode*r, int o, int m, int s);
void ldr(struct bnode* r);
int main() {

    struct dnode* t;

    // coada vida
    front = new_dnode(NULL,0);
    rearl = new_dnode(NULL,0);
    rearr = new_dnode(NULL,0);

    front->left=rearl;
    front->right=rearr;

    // insereaza apel primit
    if (cautar(front, rearr, 1123)) add_l_dlist(rearr, NULL, 1123);
    // adauga moment de timp - apel primit
    build_abc(front->right->info,10,20,0);
    // insereaza apel efectuat
    if (cautal(front, rearl, 5523)) add_r_dlist(rearl, NULL, 5523);
    // adauga moment de timp - apel efectuat
    build_abc(front->left->info,20,10,0);

    // cauta la dreapta
    t = cautar(front, rearr, 123);
    // afiseaza info abc
    ldr(t->info);
    // cauta la stinga
    t = cautal(front, rearl, 123);
    // afiseaza info abc
    ldr(t->info);

    return 0;
}

struct dnode* new_dnode(struct bnode* r, int a)
{
    struct dnode* p;
    p=(struct dnode*)malloc(sizeof(struct dnode));
    p->info = r;
    p->nr_tel=a;
    p->left = NULL;
    p->right = NULL;
    return p;
}

void add_r_dlist( struct dnode* p, struct bnode* r, int a)
{
    struct dnode* h;
    if (p!=NULL)
    {   h=new_dnode(r, a);
        h->right=p->right;
        h->left=p;
        p->right->left=h;
        p->right=h;
    }
```

```
}

void add_l_dlist( struct dnode* p, struct bnode* r, int a)
{
    struct dnode* h;
    if (p!=NULL)
    {   h=new_dnode(r, a);
        h->left=p->left;
        h->right=p;
        p->left->right=h;
        p->left=h;
    }
}

int del_r_dlist( struct dnode* p)
{
    struct dnode* q;
    int x=-1;

    if (p!=NULL)
    {
        q=p->right;
        x = del_dlist(q);
    }
    return x;
}

int del_l_dlist( struct dnode* p)
{
    struct dnode* q;
    int x=-1;

    if (p!=NULL)
    {
        q=p->left;
        x = del_dlist(q);
    }
    return x;

}

int del_dlist( struct dnode* p)
{
    int x=-1;

    if (p!=NULL)
    {
        p->left->right=p->right;
        p->right->left=p->left;
        x= p->nr_tel;
        free(p);
    }
    return x;
}

// parcurgere dreapta
```

```
void afisarer(struct dnode* f, struct dnode* r)
{
    while (f->right != r)
    {
        printf("%d, ", f->right->nr_tel);
        f=f->right;
    }
    printf("\n");
}

// parcurgere stinga
void afisarel(struct dnode* f, struct dnode* r)
{
    while (f->left != r)
    {
        printf("%d, ", f->left->nr_tel);
        f=f->left;
    }
    printf("\n");
}

// cautare in coada la dreapta - intre front si rearr
struct dnode* cautar(struct dnode* f, struct dnode* r, int a)
{
    struct dnode* p;
    struct dnode* res = NULL;

    p=f;
    while (p->right != r)
    {
        if (a == p->right->nr_tel) {
            res=p->right;
            break;
        }
        p=p->right;
    }
    return res;
}

// cautare in coada la stinga - intre front si rearl
struct dnode* cautal(struct dnode* f, struct dnode* r, int a)
{
    struct dnode* p;
    struct dnode* res = NULL;

    p=f;
    while (p->left != r)
    {
        if (a == p->left->nr_tel) {
            res=p->left;
            break;
        }
        p=p->left;
    }
    return res;
}
```

```c
// nod nou in abc dupa timp
struct bnode* new_tree_node(int o, int m, int s)
{
    struct bnode* p;

    p= (struct bnode*) malloc(sizeof(struct bnode));
    p->ora=o;
    p->min=m;
    p->sec=s;
    p->left=NULL;
    p->right=NULL;
}

// creare nod in abc - dupa timp
struct bnode* build_abc(struct bnode*r, int o, int m, int s)
{
    int t1;
    int t2;
    if (r==NULL) r= new_tree_node(o,m,s);
    else
    {
        t1=(r->ora)*3600+(r->min)*60+r->sec;
        t2=o*3600+m*60+s;
        if (t2 < t1 ) r->left=build_abc(r->left,o, m,s);
        if (t2 > t1 ) r->right=build_abc(r->right,o,m,s);
    }

    return r;
}

// cautare nod in abc - dupa timp
struct bnode* search_abc(struct bnode*r, int o, int m, int s)
{
    int t1;
    int t2;
    t1=(r->ora)*3600+(r->min)*60+r->sec;
    t2=o*3600+m*60+s;

    if (r == NULL)  return NULL;
    if (t1 == t2) return r;
    if (t2 < t1) return (search_abc(r->left,o, m, s));
    if (t2 > t1) return (search_abc(r->right,o, m, s));

}

// parcurgere in ordine dupa timp
void ldr(struct bnode* r)
{
    if(r!=NULL)
    {
        ldr(r->left);
        printf("%.2d : %.2d : %.2d\n", r->ora, r->min, r->sec);
        ldr(r->right);
    }
}
```

Observații:

Fiecare subiect are 1 punct.

Pentru fiecare subiect se cer: descrierea logica, desene explicative si print-screen-uri din depanatorul CLion care explica funcționarea codului.

Toate cerințele sunt obligatorii.