Laborator 2:
structuri de date(i)

Întocmit de: Claudia Pârloagă

Îndrumător: Asist. Drd. Gabriel Danciu

I. NOŢIUNI TEORETICE

Agner Krarup Erlang, matematician şi inginer danez care a lucrat la Copenhagen Telephone Company, a publicat în 1909 prima lucrare privind teoria aşteptării în care sunt enunţate disciplinele FIFO, LIFO care, mai târziu, au stat la baza structurilor de date **coadă** şi **stivă**. Totodată, Erlang este unitatea de măsură pentru volumul traficului din telecomunicaţii. Stiva utilizată în arhitectura calculatorului a fost patentată în 1957 de Friedrich L. Bauer, un pioner în ştiinţa calculatoarelor, de origine germană.

A. Stiva(Stack)

Stiva este o structură de date de tip *last in-first out*(LIFO- ultimul intrat-primul iesit).

Pentru o stivă S avem:

- S[1]=elementul de la baza stivei
- top[S]=vârful stivei
- S[top[S]]=elementul din vârf
- top[S]=0 : stiva este goală
- o stivă S conţine elementele de la 1 până la top[S].
- *underflow* = încercarea de a scoate un element din stivă goală(operaţie eronată)
- *overflow* = încercarea de a introduce un element într-o stivă plină(operatie eronata)
- Operaţia de inserare a unui element în stivă se numeşte **PUSH**
- Operaţia de extragere a unui element din stivă se numeşte **POP**

Prezentăm mai jos pseudocodul pentru operaţiile asupra unei stive:

- Operaţia **PUSH**

$PUSH(S, x)$

1. $top[S] \leftarrow top[S] + 1$
2. $S[top[S]] \leftarrow x$

Figura 1: Operaţia PUSH

- Operaţia **POP**

```
POP(S)
1. if STACK − EMPTY(S) then
2.       error "underflow"
3. else   top[S] ← top[S] − 1
4. return S[top[S] + 1]
```

Figura 2: Operaţia POP

B. Coada(Queue)

Coada este o structură de date de tip *first in-first out*(FIFO: primul intrat-primul ieşit).

Pentru o coadă Q avem:

- primul element=cap(*head*)
- ultimul element=coada(*tail*)
- head[Q]= indexul primului element
- tail[Q]= indexul ultimului element
- head[Q]= tail[Q]: coadă goală
- *underflow*= încercarea de a scoate un element dintr-o coadă goală
- *overflow*= încercarea de a introduce un element într-o coadă plină
- Operaţia de inserare a unui element în coada se numeşte **ENQUEUE**
- Operaţia de extragere a unui element din coadă se numeşte **DEQUEUE**

Prezentăm mai jos pseudocodul pentru operaţiile asupra unei cozi:

- Operaţia **ENQUEUE**

```
ENQUEUE(Q, x)
1. Q[tail[Q]] ← x
2. if tail[Q] = length[Q]
3.       then tail[Q] ← 1
4.       else tail[Q] ← tail[Q] + 1
```

Figura 3: Operaţia ENQUEUE

- Operaţia **DEQUEUE**

```
DEQUEUE(Q)
1. x ← Q[head[Q]]
2. if head[Q] = length[Q]
3.       then head[Q] ← 1
4.       else head[Q] ← head[Q] + 1
5. return x
```

Figura 4: Operaţia DEQUEUE

C. Liste înlănţuite

O **listă înlănţuită** este o structură de date dinamică în care fiecare element conţine o referinţă către următorul element.

Ultimul nod din listă poate fi doar una din următoarele:

- o legătură NULL ce nu indică nici un nod
- o legătură dummz ce nu conţine informaţie validă
- conţine o legătură către primul nod(în acest caz lista va fi circulară)

1. Liste simplu înlănţuite

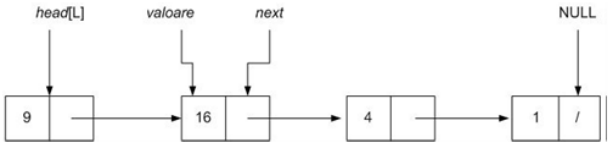

Figura 5: Listă simplu înlănţuită

Pentru o listă simplu înlănţuită L avem:

- un nod conţine valoarea acelui element şi referinţa către următorul nod din listă
- valoarea unui element este notată cu *key*
- head[L]=primul element(nod) din listă
- head[L]=NULL: listă goală
- next = referinţa către următorul element din listă

Algoritmul de **căutare într-o listă simplu înlănţuită**, după o valoare k:

$LIST - SEARCH(L, k)$

1. $x \leftarrow head[L]$
2. **while** $x \neq NULL$ **and** $key[x] \neq x$ do
3. $\quad x \leftarrow next[x]$
4. **return** x

Figura 6: Căutarea într-o listă simplu-înlănţuită

2. Liste dublu-înlănţuite

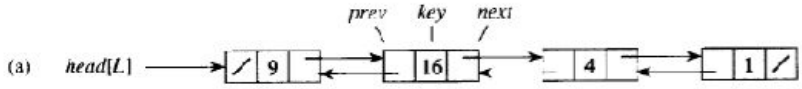

Figura 7: Listă dublu- înlănţuită

Pentru **o listă dublu-înlănţuită L** avem:

- fiecare element al listei conţine: o cheie şi două referinţe: *prev* şi *next*
- *prev*= indică nodul predecesor din listă
- *next*= indică nodul succesor din listă
- pentru un element x al listei:
  - next[x]= referinţa către succesorul nodului x
  - next[x]= NULL atunci elementul x nu are successor şi x= coada listei
  - prev[x]= referinţa către predecesorul nodului x
  - prev[x]= NULL atunci elementul x nu are predecesor şi x=capul listei(head)
  - head[L]= NULL atunci lista este goală

- Inserarea într-o listă dublu-înlănţuită

$LIST - INSERT(L, x)$

1. $next[x] \leftarrow head[L]$
2. ***if*** $head[L] \neq NULL$ ***then***
3. $\quad prev[head[L]] \leftarrow x$
4. $head[L] \leftarrow x$
5. $prev[x] \leftarrow NULL$

Figura 8: Inserarea într-o listă dublu- înlănţuită

- Ştergerea dintr-o listă dublu-înlănţuită

$LIST - DELETE(L, x)$

1. ***if*** $prev[x] \neq NULL$ ***then***
2. $\quad next[prev[x]] \leftarrow next[x]$
3. ***else***
4. $\quad head[L] \leftarrow next[x]$
5. ***if*** $next[x] \neq NULL$ ***then***
6. $\quad prev[next[x]] \leftarrow prev[x]$

Figura 9: Ştergerea dintr-o listă dublu- înlănţuită

II. PREZENTAREA LUCRĂRII DE LABORATOR

A. Stiva

Codul de mai jos exemplifică modul de implementare al unei stive:

```
1  package stiva;
2  import java.util.NoSuchElementException;
3
4  public class Stiva {
5
6      private int top;
7      private int dim;
8      private int[] stiva;
9
10     Stiva(int size)
11     {
12         if(size<=0)
13             throw new IllegalArgumentException("Dimensiunea stivei trebuie sa fie pozitiva!");
14         dim=size;
15         stiva=new int[dim];
16         top=-1;
17     }
18
19     public void push(int x)
20     {
21         if(top==stiva.length)
22             throw new NoSuchElementException("Stiva este plina");
23         top=top+1;
24         stiva[top]=x;
25
26     }
27
28     public int pop()
29     {
30         //if(top==-1)
31         if(isEmpty())
32             throw new NoSuchElementException("Stiva este goala!");
33         int temp=stiva[top];
34         top=top-1;
35         return temp;
36     }
37
38     //verifica daca stiva este goala sau nu
39     public boolean isEmpty()
40     {
41         if(top==-1)
42             return true;
43         else return false;
44     }
45
46     //afiseaza elementele stivei
47     public void display()
48     {
49         if(top==-1)
50             System.out.println("Stiva goala");
51         else for(int j=0;j<=top;j++)
52             System.out.print(stiva[j]+" ");
53     }
54
55 }
```

Clasa de testare:

```
1  package stiva;
2
3  import java.util.Scanner;
4
5  public class Main {
6
7      public static void main(String[] args) {
8
9          Scanner s=new Scanner(System.in);
10         System.out.println("Dati dimensiunea stivei: ");
11         int dim=s.nextInt();
12         Stiva stiva=new Stiva(dim);
13
14         boolean gata = true ;
15         printCommands () ;
16
17         while (gata) {
18
19             try {
20                 System.out.print ("Comanda: ") ;
21
22                 int param=s.nextInt();
23                 switch (param) {
24
25                     case 0: {
26                         gata = false ;
27                         break ;
28                     }
```

```
                case 1: {
                    System.out.println("Dati elementul de inserat");
                    int x=s.nextInt();
                    stiva.push(x);
                    break ;
                }
                case 2: {
                    System.out.println("Elementul extras este: "+stiva.pop());
                    break ;
                }
                case 3: {
                    stiva.display();
                    System.out.println();
                    break ;
                }
                case 4: {
                    printCommands () ;
                    break ;
                }

                default : {
                    System.out.println ("Invalid command") ;
                    break ;
                }
            }
        }
        catch (Exception e) {
            System.out.println ("Invalid operation, program exiting." + e.getMessage ()) ;
            System.exit (0) ;
        }
    }
}

private static void printCommands () {
    System.out.println ("1: insereaza element(push)") ;
    System.out.println ("2: extrage" +
            " element(pop)") ;
    System.out.println ("3: afiseaza stiva") ;
    System.out.println ("4: afiseaza lista de optiuni") ;
    System.out.println ("0: Exit") ;
}

}
```

B. Coada

Codul de mai jos exemplifică modul de implementare al unei cozi:

```
package coada;

import java.util.NoSuchElementException;

public class Coada {

    private int[] coada;
    private int dim; //dimensiunea cozii
    private int head;
    private int tail;

    public Coada(int size)
    {
        if(size<=0)
            throw new IllegalArgumentException("Dimensiunea cozii trebuie sa fie pozitiva!");
        dim=size;
        coada=new int[dim];
        head=0;
        tail=-1;
    }

    public void enqueue(int x)
    {
        if(tail>=dim-1)
                    throw new NoSuchElementException("Coada este plina!");
        tail=tail+1;
        coada[tail]=x;
    }

    public int dequeue()
    {
        int temp=coada[head];
        if(move()<-1) throw new NoSuchElementException("Coada goala!!!");
        tail--;
        return temp;

    }

    //metoda realizeaza mutarea spre stanga a elementelor cozii
    public int move()
    {
        //verific daca coada este goala
        if(tail<head)
            throw new NoSuchElementException("Coada este goala!");

        int res=-1;
```

```
47
48          //mut spre stanga
49          for(int i=head;i<tail;i++)
50          {
51              coada[i]=coada[i+1];
52              res++;
53          }
54
55          //retin intr-o variabila faptul ca am executat mutarile
56          return res;
57      }
58
59      //afisarea cozii
60      public void display()
61      {
62          if(head>tail)
63              System.out.println("Coada goala!");
64          else for(int j=head;j<=tail;j++)
65              System.out.print(coada[j]+" ");
66      }
67
68  }
```

Clasa de testare:

```
1   package coada;
2   import java.util.Scanner;
3
4   public class Main {
5
6       public static void main(String[] args) {
7
8           Scanner s=new Scanner(System.in);
9           System.out.println("Dati dimensiunea cozii: ");
10          int dim=s.nextInt();
11          Coada coada=new Coada(dim);
12
13          boolean gata = true ;
14          printCommands () ;
15
16          while (gata) {
17
18              try {
19                  System.out.print ("Comanda: ") ;
20
21                  int param=s.nextInt();
22                  switch (param) {
23
24                      case 0: {
25                          gata = false ;
26                          break ;
27                      }
28                      case 1: {
29                          System.out.println("Dati elementul de inserat");
30                          int x=s.nextInt();
31                          coada.enqueue(x);
32                          break ;
33                      }
34                      case 2: {
35                          System.out.println("Elementul extras este: "+coada.dequeue());
36                          System.out.print("Coada dupa extragere: ");
37                          coada.display();
38                          break ;
39                      }
40                      case 3: {
41                          System.out.print("Coada este: ");
42                          coada.display();
43                          System.out.println();
44                          break ;
45                      }
46                      case 4: {
47                          printCommands () ;
48                          break ;
49                      }
50
51                      default : {
52                          System.out.println ("Invalid command") ;
53                          break ;
54                      }
55                  }
56              }
57              catch (Exception e) {
58                  System.out.println ("Invalid operation, program exiting." + e.getMessage ()) ;
59                  System.exit (0) ;
60              }
61          }
62      }
63
64      private static void printCommands () {
65          System.out.println ("1: insereaza element(enqueue)") ;
66          System.out.println ("2: extrage element(dequeue)") ;
67          System.out.println ("3: afiseaza coada") ;
68          System.out.println ("4: afiseaza lista de optiuni") ;
69          System.out.println ("0: Exit") ;
70      }
71
72  }
```

III. TEMĂ

Pentru fiecare din următoarele probleme se va scrie pseudocodul şi apoi codul în Java corespunzător.

Problema 1:

Implementaţi o stivă folosind o listă simplu înlănţuită(singly linked list) L. Operaţiile PUSH şi POP vor trebui să aibă complexitatea $O(1)$.

Problema 2:

Implementaţi o coadă folosind o listă simplu înlănţuită L. Operaţiile ENQUEUE şi DEQUEUE ar trebui să aibă complexitatea $O(1)$.

Problema 3:

Implementaţi operaţiile INSERT, DELETE şi SEARCH corespunzătoare unui dicţionar folosind liste circulare simplu înlănţuite.

Problema 4:

Operaţia dinamică UNION(reuniunea) preia 2 mulţimi disjuncte S_1 şi S_2 şi returnează o mulţime $S = S_1 \cup S_2$ ce conţine toate elementele din S_1 şi S_2.
Implementaţi operaţia de reuniune folosind o structură de date de tip listă potrivită într-un timp de $O(1)$.

Problema 5:

Scrieţi o aplicaţie care uneşte 2 liste simplu înlănţuite, sortate într-o singură listă simplu înlănţuită sortată, fără a folosi santinele.
Apoi, scrieţi o aplicaţie similară folosind o santinelă cu cheia ∞ pentru a marca sfârşitul fiecărei liste.
Comparaţi simplitatea codului celor 2 aplicaţii.

Problema 6:

Scrieţi o funcţie nonrecursivă care inversează o listă simplu înlănţuită de n elemente.