Tema 24 – TAD ListăOrdonată

Implementare folosind o listă dublu înlănțuită cu înlănțuirile reprezentate pe tablou

Problemă

Se dă o listă de procese caracterizate prin nume, utilizare procesor (1-100%) şi timp de rulare şi numele unui proces din listă. Acestea vor fi preluate de către sistem si rulate în funcţie de factorul de utilizare al procesorului (cele cu factor mai mic au prioritate). Să se afişeze după cât timp procesul cu numele dat va fi preluat de către sistem.

TAD ListăOrdonată

$L = \{l \mid l = [e_1, e_2, ..., e_n], e_i \in T \text{ Comparabil şi } e_1 \leq e_2 \leq ... \leq e_n, \forall i = 1, 2, . . . , n\}$

("≤" este o relaţie de ordine generică)

Interfață

• creează (l)
pre : true
post : $l \in L$, $l = \Phi$ lista vidă

• adaugă (l, e)
pre : $l \in L$, $e \in$ T Comparabil
post : $l` = (e_1, . . . , e_{i-1}, e, e_{i+1}, . . . , e_n)$ $(e_{i-1} <= e <= e_{i+1})$

• şterge (l, i, e)
pre : $l \in L$, $l = (e_1, . . . , e_{i-1}, e_i, e_{i+1}, . . . , e_n)$, $i \in$ T Întreg, i poziţie validă
post : $e \in$ T Comparabil, e = elementul de pe poziția i din l
$l` = (e_1, . . . , e_{i-1}, e_{i+1}, . . . , e_n)$
@ aruncă excepţie dacă i nu e valid

• caută (l, e)

pre : $l \in L$, $e \in T$ Comparabil

post : $\text{caută} = \begin{cases} i, \text{dacă i e prima poziție pe care e a fost găsit în lista l} \\ -1, e \notin L \end{cases}$

• element (l, i, e)

pre : $l \in L$, $i \in T$ Întreg, i poziție validă

post : $e \in T$ Comparabil, e = elementul de pe poziția i din l

@ aruncă excepție dacă i nu e valid

• vidă (l)

pre : $l \in L$

post : $\text{vidă} = \begin{cases} \text{true}, \text{dacă } l = \Phi \\ \text{false}, \text{altfel} \end{cases}$

• dim (l)

pre : $l \in L$

post : dim = $n \in T$ Întreg,

n = numărul de elemente din lista l

• iterator (l, i)

pre : $l \in L$

post : $i \in I$, i este un iterator pe lista l

• distruge (l)

pre : $l \in L$

post : l a fost "distrusă" (spațiul de memorie alocat a fost eliberat)

TAD IteratorListăOrdonată

I = {i | i este un iterator pe o listă ordonată având elemente de tip T Comparabil}

Interfață

• creează (i, l)

pre : l este o listă ordonată

post : $i \in I$, s-a creat iteratorul i pe lista ordonată l

(*curent* din iterator referă "primul" element din listă)

• element (i, e)

pre : i ∈ I, *curent* din iterator este valid (referă un element din listă)

post : e ∈ T Element, e este elementul curent din iterație

(elementul din listă referit de *curent*)

• valid (i)

pre : i ∈ I

post : $valid = \begin{cases} \text{adevărat, dacă } \textit{curent} \text{ referă o poziție validă din listă} \\ \text{fals, altfel} \end{cases}$

• următor (i)

pre : i ∈ I, *curent* este valid

post : *curent*` referă "următorul" element din listă față de cel referit de *curent*

• distruge (i)

pre : i ∈ I

post : i a fost "distrus" (spațiul de memorie alocat a fost eliberat)

Reprezentare

```
TNod:
   • data:          TComparabil
   • prev:          TÎntreg
   • next:          TÎntreg
TIteratorListăOrdonată:
   • current:       TÎntreg
   • list:          ↑TListăOrdonată
TListăOrdonată:
   • head:          TÎntreg
   • tail:          TÎntreg
   • size:          TÎntreg
   • capacity:      TÎntreg
   • is_free:       TBool[]
   • link_table:    TNod[]
```

Operații

```
TIteratorListăOrdonată:
|    Subalgoritm creează(i, l):
|    |    i.list <- l;
|    |    i.current <- l.head;
|    SfârșitSubalgoritm
|
|    Subalgoritm element(i, e):
|    |    Dacă !i.valid() atunci:
|    |    |    @aruncă eroare;
|    |    SfârșitDacă
|    |    e <- i.list.link_table[current].data;
|    SfârșitSubalgoritm
|
|    Subalgoritm valid(i):
|    |    valid <- current != -1;
|    SfârșitSubalgoritm
|
|    Subalgoritm următor(i):
|    |    Dacă !i.valid() atunci:
|    |    |    @aruncă eroare;
|    |    SfârșitDacă
|    |    i.current = i.list.link_table[current].next;
|    SfârșitSubalgoritm
|
|    Subalgoritm distruge(i):
|    SfârșitSubalgoritm
Sfârșit

TListăOrdonată:
|    Subalgoritm creează(l, c):
|    |    l.capacity <- c;
|    |    l.size <- 0;
|    |    l.head <- -1;
|    |    l.tail <- -1;
|    |    l.is_free <- alocă(l.is_free, c, ADEVARAT);
|    |    l.link_table <- alocă(l.link_table, c, NIL);
|    SfârșitSubalgoritm
```

```
|    Subalgoritm adaugă(l, e):
|    |    Dacă l.size == l.capacity atunci:
|    |    |    @aruncă eroare;
|    |    SfârșitDacă
|    |    pozitie <- @caută loc liber in l.link_table;
|    |    l.link_table[pozitie] <- TNod(e, -1, -1);
|    |    l.is_free[pozitie] <- FALS;
|    |    l.size <- l.size + 1;
|    |    @actualizează legăturile din l;
|    SfârșitSubalgoritm
|
|    Subalgoritm șterge(l, i, e):
|    |    Dacă i invalid atunci:
|    |    |    @aruncă eroare;
|    |    SfârșitDacă
|    |    p <- @caută poziția în l.link_table a elementului nr i;
|    |    l.is_free[p] <- ADEVARAT;
|    |    l.size <- l.size - 1;
|    |    @actualizează legăturile din l;
|    |    șterge <- l.link_table[p].data;
|    SfârșitSubalgoritm
|
|    Subalgoritm caută(l, e):
|    |    current <- l.head;
|    |    position <- 0;
|    |    CâtTimp current != -1 execută:
|    |    |    Dacă l.link_table[current].data == e atunci:
|    |    |    |    caută <- position;
|    |    |    SfârșitDacă
|    |    |    current <- l.link_table[current].next;
|    |    |    position <- position + 1;
|    |    SfârșitCâtTimp
|    |    caută <- -1;
|    SfârșitSubalgoritm
|
|
|
|
|
```

```
|    Subalgoritm element(l, i, e):
|    |    Dacă i invalid atunci:
|    |    |    @aruncă eroare;
|    |    SfârșitDacă
|    |    p <- @caută poziția în l.link_table a elementului nr i;
|    |    e <- l.link_table[i];
|    SfârșitSubalgoritm
|
|    Subalgoritm vidă(l):
|    |    vidă <- l.size == 0;
|    SfârșitSubalgoritm
|
|    Subalgoritm dim(l):
|    |    dim <- l.size;
|    SfârșitSubalgoritm
|
|    Subalgoritm iterator(l, i):
|    |    i <- TiteratorListăOrdonată(l);
|    SfârșitSubalgoritm
|
|    Subalgoritm distruge(l):
|    |    dealocă(l.is_free);
|    |    dealocă(l.link_table);
|    SfârșitSubalgoritm
Sfârșit
```

Complexități

Iterator:

- creează -> Θ(1)
- element -> Θ(1)
- valid -> Θ(1)
- următor -> Θ(1)
- distruge -> Θ(1)

Listă:

- creează -> Θ(n)
- adaugă -> Θ(n)
  - Caz favorabil: O(1) atunci când prima poziție din tabela de înlănțuiri este liberă, indiferent de numărul de elemente din listă
  - Caz defavorabil: O(n) atunci când ultima poziție din tabela de înlănțuiri este liberă, indiferent de numărul de elemente din listă
  - Caz mediu: Θ(n), $\sum_{k=1}^{p} 1 \in \Theta(n)$, unde p este prima poziție liberă din tabela de înlănțuiri
- șterge -> Θ(n)
  - Caz favorabil: O(1) atunci când parametrul i este 0
  - Caz defavorabil: O(n) atunci când i este lungimea listei - 1
  - Caz mediu: Θ(n), $\sum_{k=1}^{i} 1 \in \Theta(n)$
- caută -> Θ(n)
- element -> Θ(n)
- vidă -> Θ(1)
- dim -> Θ(1)
- iterator -> Θ(1)
- distruge -> Θ(1)

Diagramă apeluri