Luca Laura-Claudia

914

ADT Sparse Matrix

Domain:

SM = { sm | sm is a sparse matrix with elements of type TElem, each elements having a unique position determined by a line and a column (both integers) in sm }

Operations:

- **init(sm, nrL, nrC)**
  Pre: nrL ∈ integer, nrC ∈ integer
  Post: sm ∈ SM, sm is an "empty" matrix (only contains $0_{\textbf{TElem}}$ values)

- **nrLines(sm)**
  Pre: sm ∈ SM
  Post: nrLines <- number of lines (nrL)

- **nrColumns(sm)**
  Pre: sm ∈ SM
  Post: nrColumns <- number of columns (nrC)

- **element(sm, i, j)**
  Pre: sm ∈ SM, i ∈ TLine , j ∈ TColumn, valid(i, j)
  Post: e ∈ TElem
  element <- e ∈ (i, j) (the element on line i and column j)
  *@throws exception **if** the pair (i, j) is not valid*
- **modify(sm, i, j, val)**
  Pre: sm ∈ SM, i ∈ TLine , j ∈ TColumn, val ∈ TElem, valid(i, j)
  Post: the element from position (i, j) = val
  *@throws exception **if** the pair (i, j) is not valid*

- **destroy(sm)**
  Pre: sm ∈ SM
  Post: sm was destroyed (allocated memory was freed)

- **iterator(sm, i)**
  Pre: sm ∈ SM
  Post: i ∈ IteratorSM

Representation :

SMElement:

line: TLine

column: TColumn

value: TElem

SMNode:

Info : SMElement

Next: ↑SMNode

SM:

elems: ↑SMNode [ ] //an array of pointers to nodes

m: Integer

h:TFunction //the hash function

hc: TFunction //The hashCode function

nrL: Integer

nrC: Integer

Iterator SM

Operations:

- **init(it,sm):**
  Pre: sm $\in$ SM
  Post: it $\in$ IteratorSM
- **getCurrent(it,elem)**
  Pre: it $\in$ IteratorSM, the iterator is valid
  Post: elem $\in$ SMElement
- **next(it)**
  Pre: it $\in$IteratorSM, the iterator is valid
  Post: goes to the next element
- **valid(it)**
  Pre: it $\in$IteratorSM
  *Post: checks **if** the current element is valid*

Representation:

IteratorSM:

sm:Sparse Matrix

pos: intreger

findMin:TFunction

ADT Sparse Matrix – Operation implementation

subalgoritm init (sm, nrL, nrC) is :

    sm.m←10
    @create an array empty with m positions in sm.elems
    for i←1,m,1 **execute**:
        sm.elems[i]←NIL
    end-for
    sm.nrL←nrL
    sm.nrC←nrC

Complexity: θ(m) , is the size of the array

function nrLines(sm) is:

    nrLines← sm.nrL

Complexity: θ(1)

function nrColumns(sm) is:

    nrColumns←sm.nrC

Complexity: θ(1)

function element (sm, i, j) is:

    if i < sm.nrL or i > sm.nrL or j<sm.nrC or j>sm.nrC **then**
    @throw exception
    else
        key←sm.hc(i, j)
        position←sm.h(key)
        current←sm.elems[position]
        while currentN ≠ NIL **and not (**[currentN].line = i and [currentN].column =)j **execute**
            current←[currentN].next
        end-while
        if currentN ≠NIL **then**
            element← [currentN].info
        else
            element ←0
        end-if
    end-if

Complexity:

Best Case: the current element is NIL => *θ(1)*

Worst Case: the element is on the last position of the SLL
=> θ(n), n is the number of elements in the SLL

Average Case: *O(n)*

Final Complexity: O(n)

```
subalgoritm modify(sm, i, j , val) is:
    if i< sm.nrL or i>sm.nrL or j<sm.nrC or j>sm.nrC then
    @throw exception
    end-if
    if sm.element( i , j) = 0 then
        if val ≠ 0 then
            modifyZN(sm,i,j,val)
        end-if
    else
        if val = 0 then
            modifyNZ( sm,i,j,val)
        end-if
    end-if
```

Complexity: O(n)

```
subalgorithm modifyNZ(sm,I,j,val) is:
    key←sm.hc(I,j)
    position←sm.h(key)
    current←sm.elems[position]
    while [current].next ≠ NIL and not
      ([[current].next].info.line = i and [[current].next].info.column =  j ) execute:
        current←[current].next
    end-while
    if  [current].next = NIL then
        deallocate(sm.elems[position])
        sm.elems[position]←NIL
    else
        node←[current].next
        [current].next←[[current].next].next
        deallocate(node)
    end-if
```

Complexity: O(n) ,n is the number of elements in the SLL

```
subalgorithm modifyZN(sm,I,j,val) is:
    key←sm.hc(I,j)
    position←sm.h(key)
    @create new SMElement in elem
    [elem].info.line←i; [elem].info.column←j; [elem].info.val←val;[elem].next←NIL
    if sm.elems[position] = NIL then
        Sm.elems[position]←elem
    else
        current←sm.elems[position]
        while [current].next ≠ NIL and not ( [[current].next].info.line > [elem].info.line or
    [[current].next].info.line =  [elem].info.line) execute:
            current←[current].next
```

```
        end-while
        if [current].next = NIL
            if current=sm.elems[position]
                if [current].next].info.line = i > [elem].info.line then
                    [current].next←elem
                else
                    if [current].next].info.line = i < [elem].info.line then
                        [elem].next←current
                        Sm.elems[position]<-elem
                    else
                        If [current].next].info.column = i < [elem].info.column then
                            [current].next←elem
                        else
                            [elem].next←current
                            Sm.elems[position]←elem
                        end-if
                    end-if
            else
                If [current].info.line < [elem].info.line
                    [current[.next←elem
                end-if
            end-if
    else
        If [[current].next].info.line  > [elem].info.line
            [elem].next←[current].next
            [elem].next←elem
        else
            If [[current].next].info.line  = [elem].info.line then
                If [[current].next].info.column  > [elem].info.column then
                    [elem].next←[current].next
                    [elem].next←elem
                else
                    [elem].next←[[current].next].next
                    [[current].next].next←elem
                end-if
            end-if
    end-if
```

Complexity: O(n) , n is the number of elements in the SLL

```
subalgoritm destroy(sm) is:
    for i←1,m,1 execute
        currentN←elems[i]
        while currentN≠ NIL execute:
            nextNode←[currentN].next
            free(currentN)
            currentN←nextNode
        end-while
    end-for
```

Complexity: O(m+e) , e is the number of elements in the Sparse Matrix

```
subalgorithm  iterator(sm,i) is:

    i←IteratorSM(sm)
```

Complexity: θ(1)

Iterator SM – Operation Implementation

```
subalgorithm init(sm,i) is:
    @create a new sparse matrix  in matrix with sm's attributes
    for i<-1,sm.m,1 execute:
        Nod<-sm.elems[i]
        While nod ≠  NIL execute
            @create a new SMElement in elem
            matrix.modify( [nod].info.line, [nod].info.column, [nod].next)
            Nod<-[nod].next
        end-while
    i.sm<-matrix
    Pos<-i.findMin()
```

Complexity: O(m+e) , e is the number of elements in the Sparse Matrix

```
function  findMin() is:

    minLin ←maxInt
    minCol ←maxInt
    iminPos← -1
    for k←1,sm.m, execute
        nod←i.sm.elems[k]
        if nod ≠NIL
            if [nod].info.line <minLin
                minL=[nod].info.line
                minPos<-k
            else
                if [nod].info.line = minLin and [nod].info.column < minCol then
```

minCol←[nod].info.column
minPos←k
end-if
end-if
end-for
findMin←minPos
Compelxity: θ(m)

subalgorithm next(i) is:
i.sm.elems[pos]←[i.sm.elems[pos]].next
pos←i.findMin(i)
Compelxity: θ(m)

function valid(i) **is:**
if pod= -1
valid←false
else
valid←true
Compelxity: θ(1)

subalgorithm destroy(i) is:
i.sm.destroy()
Complexity: O(m+e) , e is the number of elements in the Sparse Matrix

Problem statement:

We are given the adjacency matrix for a directed graph with n vertices and m edges, where n ∈[10, 999] and m∈(0,30], n and m are natural numbers. Find the outbound of a given vertex v.

The vertices are denoted by integer numbers from 0 to n-1.

Justification for using ADT Sparse Matrix:

Considering the fact that we are given a fairly large number of vertices, but a small number of edges, the adjacency matrix will contain a large number of 0 and a small number of 1. In order to save memory / space, we only memorize the positions (i , j) on which the value is 1.

Solution

```
function getData() is:
	@open file
	@read in nrL and nrC the number of lines and column of the matrix
	init(matrix,nrL,nrC)
	for i←1,nrL,1 execute:
		@read a line in string mline
		for j←1,nrC,1 execute:
			modify(matrix,i,j,mline[j])
	getData←matrix
```

Compelxity: O(nrL*nrC)

```
subalgorithm solution() is:
	@create an array of 100 positions in arr
	@read a number from the console in nr
	matrix←getData()
	post=0
	Iter←sm.iterator()
	while  [iter].valid() execute
		Elem←[iter].current
		If element.line = nr
			Pos←pos+1
			Arr[pos]←elem.column
		end-if
		[iter].next
	end-while
	If pos=0
		@print “No outbound”
	else
		@print the content of arr
```

Complexity: O(m*e) , e is the number of elements in the Sparse Matrix

Tests for the ADT

```
void testSM()
{
	SM matrix(1000, 1100);
	assert(matrix.getNrL() == 1000);
	assert(matrix.getNrC() == 1100);
	assert(matrix.element(2, 2) == 0);
	matrix.modify(1, 5, 5);
	assert(matrix.element(1, 5) == 5);
	matrix.modify(2, 4, 8);
	assert(matrix.element(2, 4) == 8);
	matrix.modify(2, 2, 22);
	assert(matrix.element(2, 2) == 22);
	matrix.modify(0, 4, 80);
	assert(matrix.element(0, 4) == 80);
	matrix.modify(1, 3, 222);
	assert(matrix.element(1, 3) == 222);
	matrix.modify(2, 103, 28);
	assert(matrix.element(2, 103) == 28);
	matrix.modify(2, 3, 18);
	assert(matrix.element(2, 3) == 18);
	matrix.modify(4, 3, 28);
	assert(matrix.element(2, 103) == 28);
	matrix.modify(4, 103, 18);
	assert(matrix.element(2, 3) == 18);
	matrix.modify(1, 1, 88);
	assert(matrix.element(1, 1) == 88);
	matrix.modify(1, 1, 0);
	assert(matrix.element(1, 1) == 0);
	matrix.modify(0, 8, 55);
	matrix.modify(1, 7, 8);
	matrix.modify(7, 1, 8);
	matrix.modify(2, 6, 8);
	matrix.modify(3, 5, 333);
	matrix.modify(2, 6, 0);
	assert(matrix.element(2, 6) == 0);
	matrix.modify(3, 5, 0);
	assert(matrix.element(3, 5) == 0);
	matrix.modify(0, 8, 55);
	matrix.modify(2, 6, 8);
	matrix.modify(2, 6, 0);
	assert(matrix.element(2, 6) == 0);
	matrix.modify(7, 2, 2);
	matrix.modify(8, 1, 22);
	matrix.modify(6, 3, 555);
	matrix.modify(0, 7, 44);
	matrix.modify(0, 17, 66);
	matrix.modify(0, 87, 66);
	matrix.modify(0, 97, 66);
	matrix.modify(0, 27, 66);
	matrix.modify(0, 37, 66);

	matrix.modify(0, 5, 222);
	matrix.modify(0, 15, 222);
	matrix.modify(0, 25, 222);
	matrix.modify(0, 45, 222);
	matrix.modify(0, 35, 222);


	IteratorSM* iter = matrix.iterator();
	while (iter->valid())
	{
		iter->next();
	}
	iter->destroy();
	delete iter;
	matrix.destroy();
}
```

Tests for the Iterator

```
void testIT()
{
	SM matrix{ 1000,1000 };
	matrix.modify(1, 1, 22);
	matrix.modify(0, 2, 44);
	matrix.modify(3, 103, 9);
	matrix.modify(3, 3, 88);
	IteratorSM iter(matrix);
	while (iter.valid())
	{
		iter.next();
	}
}
```