Data Structures
HW2

Sina Rostami
9822143

Fall 2020

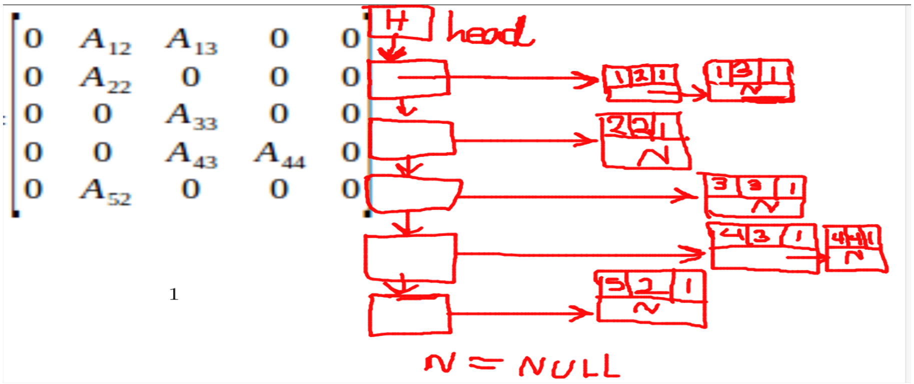
Problem 1:

Pseudo-Code

```
let linked_list be the linked-list we want to reverse.
let size be the size of the linked_list.
for i in range(0, size / 2) do
    let temp_node1 points to the head of the linked-list
    do (i) times:
        temp_node1 ← temp_node1.next
    let temp_node2 points to the head of the linked-list
    do (size - i - 1) times:
        temp_node2 ← temp_node2.next
    swap data of temp_node1 and temp_node2.
end
Now the list have reversed.
you can see the source code of the implementation in the src directory.
```

Problem 2:

we can represent the sparce matrix in linked-list representaion.
for that :
1. each Node has 2 more fields as the *i and j* index for representing where the data was in the matrix
2. each Node has pointers to its right side Node.
3. we use index nodes to represent rows of the matrix in the linked-list
for the given example matrix we have :



$$\begin{bmatrix} 0 & A_{12} & A_{13} & 0 & 0 \\ 0 & A_{22} & 0 & 0 & 0 \\ 0 & 0 & A_{33} & 0 & 0 \\ 0 & 0 & A_{43} & A_{44} & 0 \\ 0 & A_{52} & 0 & 0 & 0 \end{bmatrix}$$

Problem 3:

we can do this by followings :
1. iterate over array and push them in the stack.
2. now pop the stack (size of array) times and put them in the array.

Pseudo-Code

```
for i in the array do
    push i to the stack.
end
let index := 0.
while the stack is not empty do
    let elem := stack.pop
    put elem in the index-th cell of the array
    index ← index + 1
end
```

Problem 4:

A.

popped elements are : [h, s, f]
elements in the stack are : [d, m]

B.

dequeued elements are : [d, h, f]
elements in the queue are : [s, m]

Problem 5:

a) 2, 4, 5, 3, 1

push (1)	[∅]
push (2)	[∅]
pop	[2]
push (3)	[2]
push (4)	[2]
pop	[2, 4]
push (5)	[2, 4]
pop	[2, 4, 5]
pop	[2, 4, 5, 3]
pop	[2, 4, 5, 3, 1]

b) 1, 3, 5, 4, 2.

push (1)	[∅]
pop	[1]
push (2)	[1]
push (3)	[1]
pop	[1, 3]
push (4)	[1, 3]
push (5)	[1, 3]
pop	[1, 3, 5]
pop	[1, 3, 5, 4]
pop	[1, 3, 5, 4, 2]

Problem Bounce:

we use q_1, q_2 for implementation.
for the Push method abstraction :

```
if $q_1$.isEmpty() then
    $q_1.enqueue(element)$
end
else
    $current\_size := q_1.size()$
    for i in (0, current_size) do
        $q_2.enqueue(q_1.dequeue())$
    end
    $q_1.enqueue(element)$
    for i in (0, current_size) do
        $q_1.enqueue(q_2.dequeue())$
    end
end
```

and for the Pop method abstraction :

```
return $q_1.dequeue()$
```