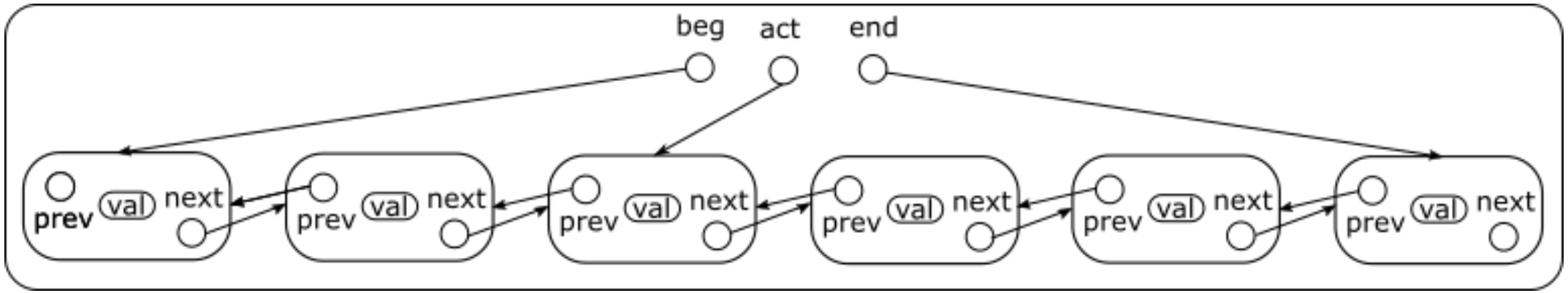
XOR linked list problem

Algorithms and Data Structures - project 2021

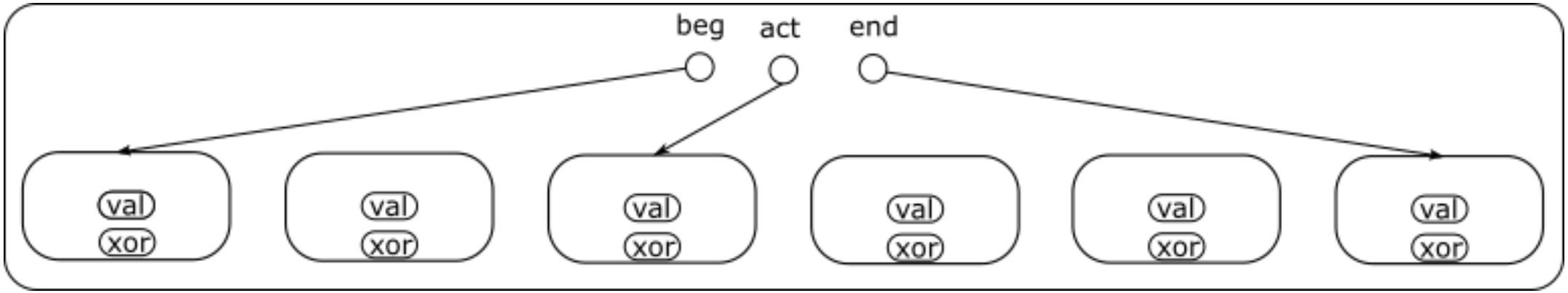
Doubly Linked List vs. XOR linked list

The task is based on implement a bidirectional list. A classic bidirectional list stores successor and predecessor addresses in nodes.



Doubly Linked List vs. XOR linked list

In our task the list stores only xor of these addresses in nodes. This saves approximately 50% of memory used by the list structure.

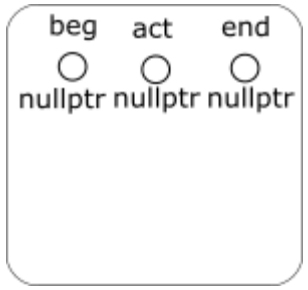


Adding elements to list (at begin or end)

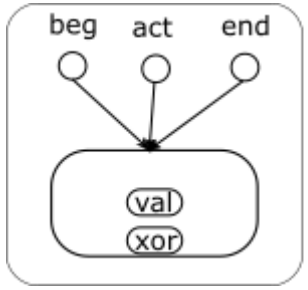
- ADD_BEG **N** - adding an element with value **N** to the beginning of the list.
- ADD_END **N** - adding an element with value **N** to the end of the list.

ADD_BEG/END 1
ADD_END 2

1. empty list

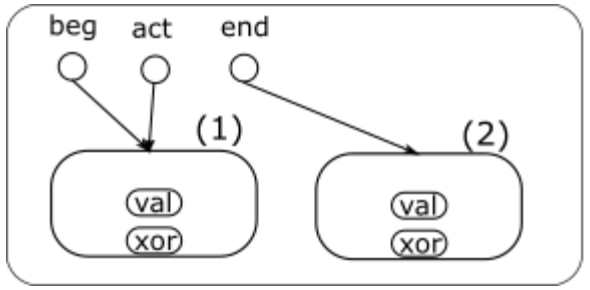


2. adding first element



xor == 0

3. add to the end



xor(1) == 0 ⊕ next == next

xor(2) == prev ⊕ 0 = prev

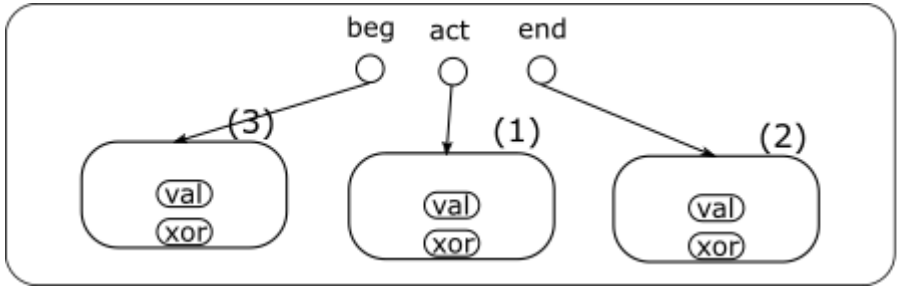
Adding elements to list (at actual)

- ADD_ACT **N** - adding an element with the value of **N** as a **predecessor** of the currently selected element (ACTUAL).

ADD_BEG 3

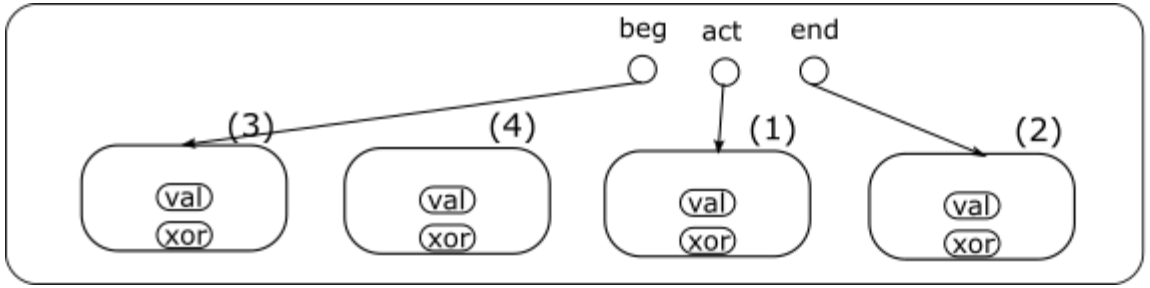
ADD_ACT 4

4. add to the beginning



xor(1) == prev ⊕ next
xor(2) == prev ⊕ 0 == prev
xor(3) == 0 ⊕ next = next

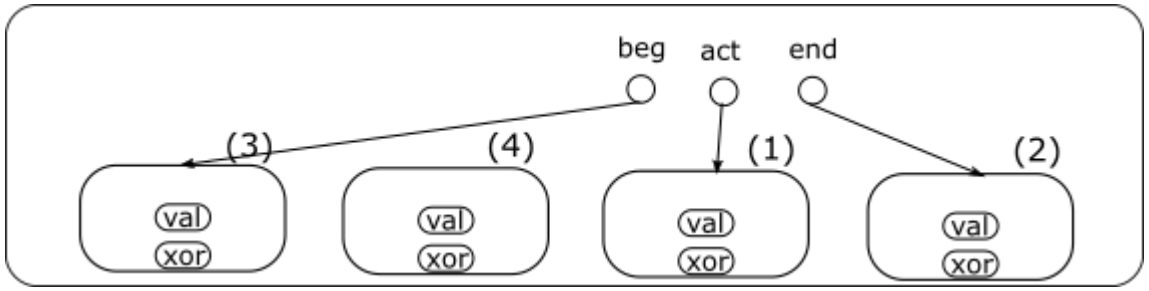
5. add to the ACTUAL



xor(1) == prev ⊕ next
xor(2) == prev ⊕ 0 == prev
xor(3) == 0 ⊕ next = next
xor(4) == prev ⊕ next

NEXT, PREV, ACTUAL commands

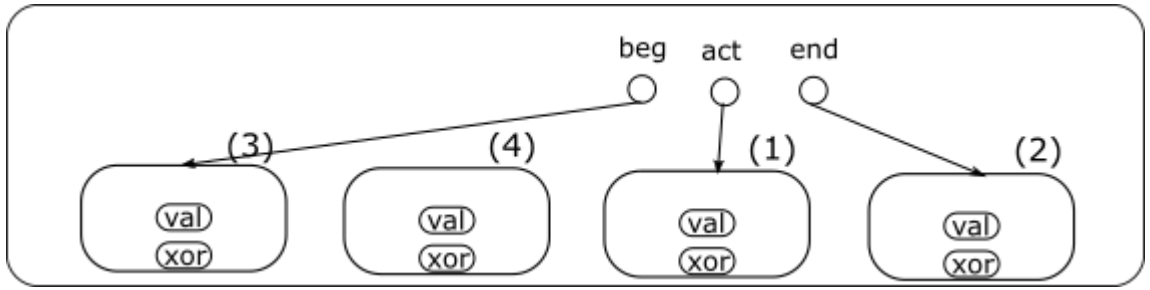
- ACTUAL - value of the currently pointed element. For an empty list it is NULL and after adding the first node to the list ACTUAL returns the value of that element as long as it is not moved by other operations.
- NEXT - prints the value of the ACTUAL successor setting it ACTUAL at the same time. If ACTUAL points to the last element of the queue, its successor will be the first element of the queue.
- PREV - prints the value of the predecessor ACTUAL setting it ACTUAL at the same time. If ACTUAL points to the first element of a queue, its predecessor will be the last element of the queue.



NEXT == 2	PREV == 4	ACTUAL == 1	ACTUAL == 1	ACTUAL == 1
PREV == 1	ACTUAL == 4	PREV == 4	NEXT== 2	PREV== 4
ACTUAL == 1	NEXT == 1	NEXT == 1	NEXT == 3	PREV == 3
PREV == 4	NEXT == 2	NEXT == 2	NEXT == 4	PREV == 2

PRINTING commands

- PRINT_FORWARD - prints the contents of the list from the first to the last element.
- PRINT_BACKWARD - prints the contents of the list from the last to the first element.

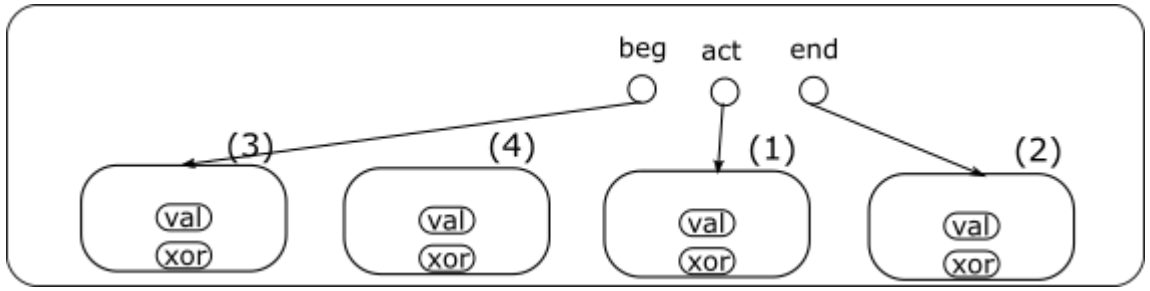


PRINT_FORWARD – 3 4 1 2

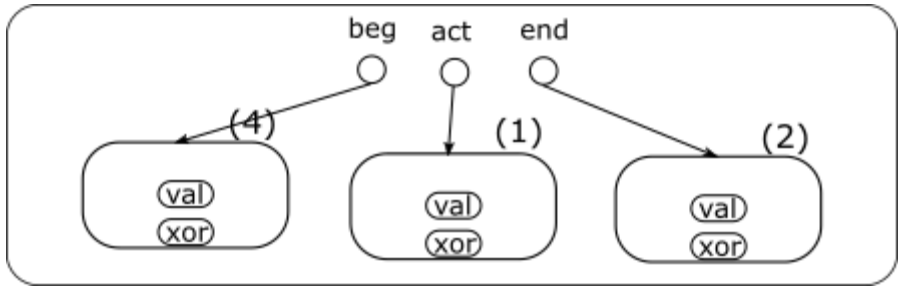
PRINT_BACKWARD – 2 1 4 3

Deleting elements from list (at begin or end)

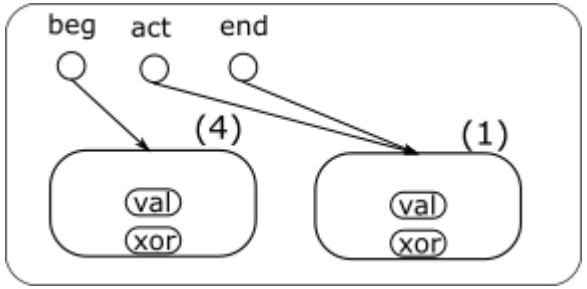
- DEL_BEG - removes the first node from the list.
- DEL_END - removes the last node from the list.



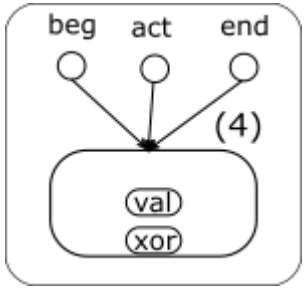
DEL_BEG



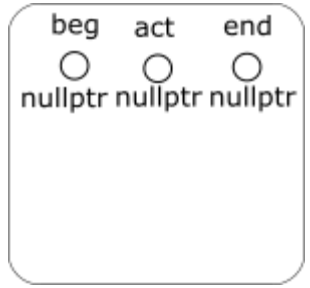
DEL_END



DEL_END

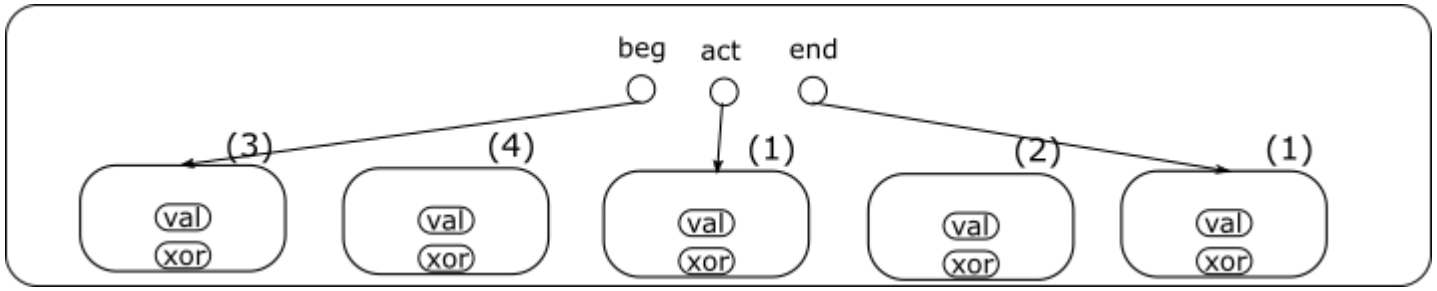


DEL_END

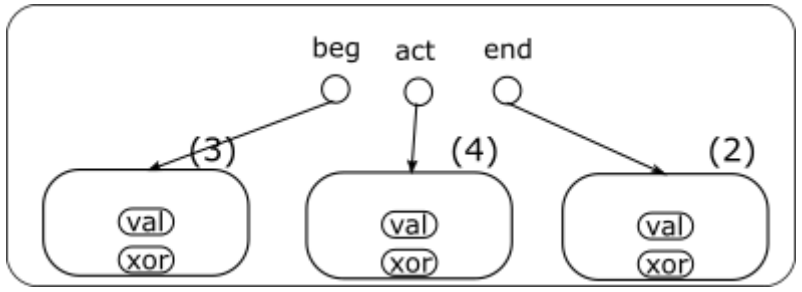


Deleting elements from list (with **N** value or actual)

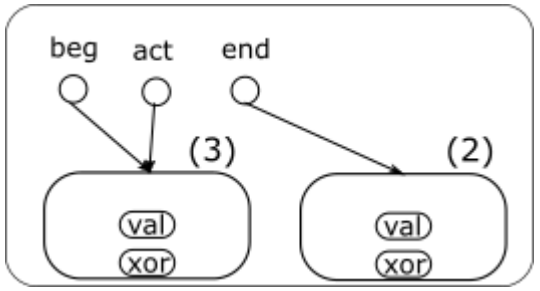
- DEL_VAL **N** - removes from the list **all** nodes whose value is equal to **N**.
- DEL_ACT - removes from the list the node that ACTUAL points to, simultaneously setting ACTUAL to PREV. In the case of the PREV does not exist (ACTUAL was the first element of the list) ACTUAL shows the last element of the list.



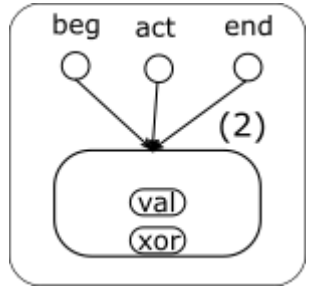
DEL_VAL **1**



DEL_ACT



DEL_ACT



Deleting elements exceptions

- DEL_BEG, DEL_END, DEL_VAL, and DEL_ACT commands for an empty list do not remove anything.
- In each of these cases, removing the currently pointed element (ACTUAL command) should result in moving the currently pointed element pointer to the preceding element, and if it does not exist, to the last element of the list.

Have fun, see you later