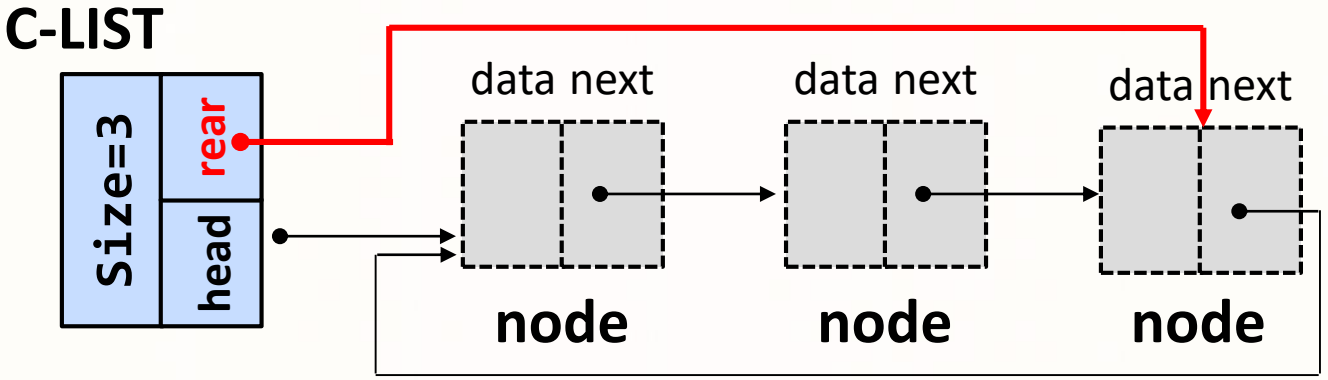
Διάλεξη 08: Λίστες ΙΙ – Κυκλικές Λίστες

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

- Κυκλικές Απλά Συνδεδεμένες Λίστες
- Κυκλικές Διπλά Συνδεδεμένες Λίστες
- Τεχνικές Μείωσης Χώρου

Κυκλικές Απλά Συνδεδεμένες Λίστες

- Μια κυκλική λίστα μπορεί να παίξει το ρόλο της ουράς, της οποίας και τα δύο άκρα είναι προσιτά με τη βοήθεια ενός μόνο δείκτη.

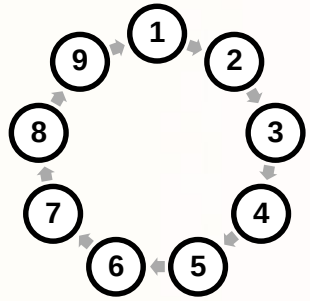


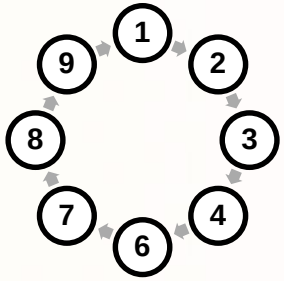
- Το πίσω άκρο, δηλαδή το άκρο όπου γίνονται οι εισαγωγές, είναι αυτό που δείχνεται από τον δείκτη στη λίστα (rear).
- Άρα οι εξαγωγές γίνονται ακριβώς μετά από τον κόμβο που δείχνεται από τον δείκτη rear.

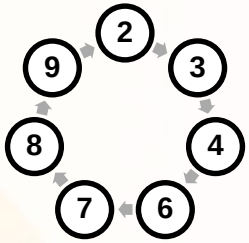
Το πρόβλημα του Josephus

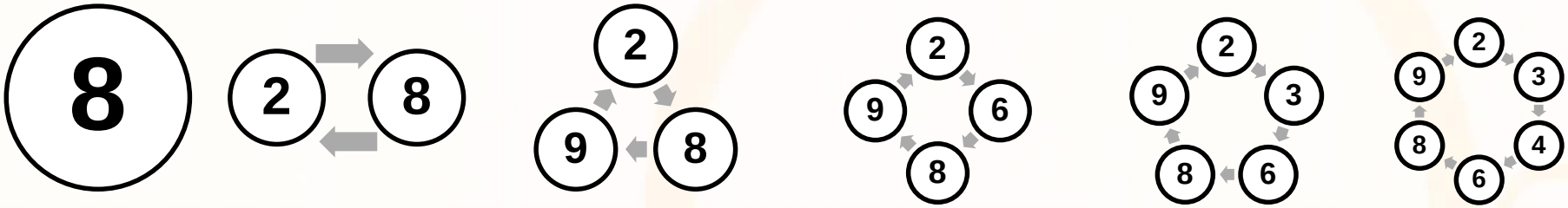
- Titus Flavius Josephus (37 – 100 μ. Χ.)
- Ν άτομα τοποθετούνται σε έναν κύκλο
- Αφαιρούμε το Μ-οστό άτομο, για κάποιο δοσμένο Μ (ξεκινώντας από το 1)
- Ξεκινώντας από Μ+1 αφαιρούμε πάλι το Μ-οστό άτομο
- κ.ο.κ. μέχρι να μείνει μόνο ένα άτομο
- Josephus(N,M): το άτομο που θα μείνει στο τέλος

Josephus(9,5)









Κυκλικά Απλά Συνδεδεμένης Λίστα (συν.)

- Άμεση λύση με χρήση κυκλικής λίστας
  - Κατασκευή λίστας Ν κόμβων
  - Ο τελευταίος κόμβος δείχνει στον πρώτο
  - Διατρέχουμε τη λίστα μέχρι να αδειάσει
    - Κυκλική διάσχιση χωρίς ειδικό κώδικα
  - Διαγράφουμε το Μ-οστό στοιχείο κάθε φορά
    - Εύκολη αφαίρεση στοιχείων ακολουθώντας δείκτες

- Δεν υπάρχει αλλαγή στις δομές (κλάσεις), μόνο στις συναρτήσεις διαγραφής και εισαγωγής

- Αναθέτουμε στον τελευταίο κόμβο να δείχνει στον πρώτο

Το πρόβλημα του Josephus: Υλοποίηση

```java
package epl231;

public class Josephus {
  public static void main(String[] args) {
   int N = Integer.parseInt(args[0]);
   int M = Integer.parseInt(args[1]);
   JosephusNode head = new JosephusNode(1);
   JosephusNode cur = head; // Head Node
   for (int i = 2; i <= N; i++) {
      JosephusNode newNode = new JosephusNode(i);
      cur.setNext(newNode);
      cur = newNode;
   }
   cur.setNext(head); // Last node points to beginning
   while (cur != cur.getNext()) {
      for (int i = 1; i < M; i++)
          cur = cur.getNext();
      System.out.println("Bye Bye " + cur.getNext().getValue());
      cur.setNext(cur.getNext().getNext()); // Farewell my friend
   }
   System.out.println("Josephus Survivor is " + cur.getValue());
  }
}
```

```java
package epl231;

public class JosephusNode {
 private int val;
 private JosephusNode next;

 JosephusNode(int v) {
   val = v;
   next = null;
 }

 int getValue() { return val; }
 void setValue(int v) { val = v;}

 JosephusNode getNext() { return next;}
 void setNext(JosephusNode n) { next = n;}
}
```

Παραδείγματα εκτέλεσης

N=9

M=5

```
Bye Bye 5
Bye Bye 1
Bye Bye 7
Bye Bye 4
Bye Bye 3
Bye Bye 6
Bye Bye 9
Bye Bye 2
Josephus Survivor is 8
```

N=1024

M=2

```
Bye Bye 2
Bye Bye 4
...

Bye Bye 641
Bye Bye 897
Bye Bye 257
Bye Bye 769
Bye Bye 513
Josephus Survivor is 1
```

N=1055 = 1024 + **31**

M=**2**

```
Bye Bye 2
Bye Bye 4
...

Bye Bye 703
Bye Bye 959
Bye Bye 319
Bye Bye 831
Bye Bye 575
Josephus Survivor is 63
```

Josephus

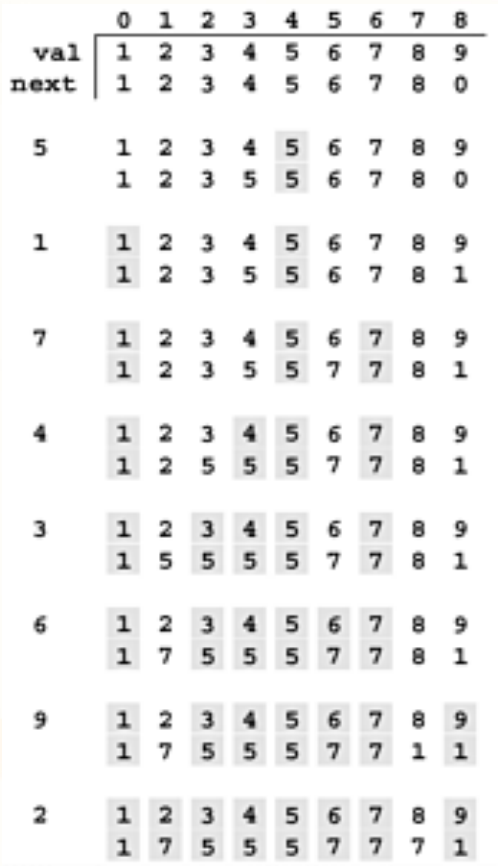

ΑΝΤ1
- ΕΣΥ, ΧΛΑΠΑΤΣΑ ΘΑ ΠΑΙΞΕΙΣ ΤΟΝ ΠΡΟΔΟΤΗ
- ΕΥΧΑΡΙΣΤΩ, ΑΡΧΗΓΕ. ΠΑΝΤΑ ΗΘΕΛΑ ΝΑ ΠΡΟΔΩΣΩ ΤΗΝ ΠΑΤΡΙΔΑ ΜΟΥ

Josephus

Το πρόβλημα του Josephus: Υλοποίηση (συν.)

- Γενικά μπορούμε να υλοποιούμε λίστες με πίνακες (δεν είναι πάντα βολικό όμως)

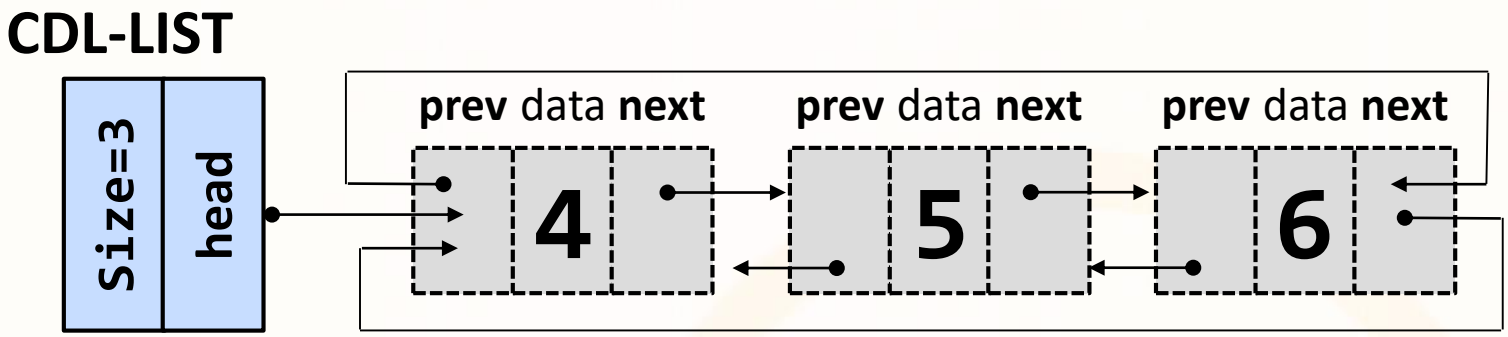
- Χρειαζόμαστε 2 πίνακες:
  - `val[i]`: στοιχείο κόμβου i
  - `next[i]`: δείκτης επόμενου κόμβου
  - Διαγραφή κόμβου γίνεται με ενημέρωση του `next[]`
  - `next[x] = next[next[x]]`



Κυκλικές Διπλά Συνδεδεμένες Λίστες

- Κυκλική λίστα της οποίας ο κάθε κόμβος δείχνει και στον επόμενο και στον προηγούμενο.

- Με την εισαγωγή κεφαλής σε τέτοιες λίστες παίρνουμε τις κυκλικές διπλά συνδεδεμένες λίστες με κεφαλή.



Παρατηρήσεις

1. Το τι είδους λίστα θα χρησιμοποιήσετε εξαρτάται πάντα από το πρόβλημα
2. Συλλογή σκουπιδιών
   - Στη Java δεν χρειάζεται να απελευθερώνουμε τη μνήμη όταν σταματάμε να χρησιμοποιούμε κάποιο δείκτη. Γίνεται αυτόματα από τον Garbage Collector
   - Σε άλλες γλώσσες πρέπει να γίνει ρητά (στη C++ καλώντας την `delete`).
   - Προσοχή στη χρήση της μνήμης!
3. Για κάθε τύπο λίστας, μπορούμε να ορίσουμε μία κλάση που να περιέχει όλες τις χρήσιμες μεθόδους ως μέλη
   - Κάνει πιο εύκολο τον κώδικα για προγράμματα-πελάτες. Οι έλεγχοι γίνονται από τις μεθόδους της κλάσης

Τεχνικές Μείωσης Χώρου

- Οι διπλά συνδεδεμένες λίστες έχουν το μειονέκτημα πως απαιτούν την ύπαρξη δύο πεδίων δεικτών σε κάθε κόμβο.
- Πολλές από τις αποθηκευμένες πληροφορίες επαναλαμβάνονται: κάθε δείκτης αποθηκεύεται σε δυο κόμβους, τον επόμενο και τον προηγούμενο με αποτέλεσμα τη σπατάλη χώρου.

Τεχνικές Μείωσης Χώρου

- *Ορισμός* (Exclusive or): $a \oplus b = 0 \Leftrightarrow a = b$, δηλ.

a	b	a ⊕ b
1	1	0
1	0	1
0	1	1
0	0	0

(a ⊕ b) ⊕ a == b	(a ⊕ b) ⊕ b == a
1	1
0	1
1	0
0	0

- Αν $a_1a_2\ldots a_n$, $b_1b_2\ldots b_n$ είναι συμβολοσειρές και $a_i \oplus b_i = c_i$ τότε ορίζουμε
- *Ιδιότητες του Exclusive-or*

  $a \oplus b = b \oplus a$ $\qquad$ $a \oplus (b \oplus c) = (a \oplus b) \oplus c$

  $a \oplus a = 0$ $\qquad$ $a \oplus 0 = a$

Τεχνικές Μείωσης Χώρου

Μέθοδος μείωσης χώρου

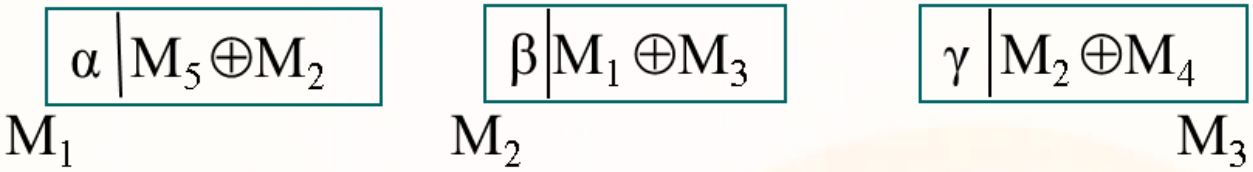
- Θεωρούμε ότι κάθε κόμβος της λίστας αποτελείται από δύο πεδία: το πεδίο `data` για αποθήκευση δεδομένων και το πεδίο `Link`. Ας υποθέσουμε πως η λίστα χρησιμοποιείται για την αποθήκευση n δεδομένων.
  - M_i: η διεύθυνση του κόμβου που περιέχει το i-οστό δεδομένο.
  - Θεωρούμε επίσης δύο επιπλέον κόμβους με διευθύνσεις M_{n+1} και M_{n+2}.
- Για κάθε κόμβο, το πεδίο `Link` περιέχει το exclusive-or της διεύθυνσης του κόμβου που προηγείται και της διεύθυνσης αυτού που ακολουθεί. Δηλαδή

$$(M_i).Link = M_{(i-1) \bmod (n+2)} \oplus M_{(i+1) \bmod (n+2)}.$$

- Για τη διέλευση της λίστας απαιτούνται δύο δείκτες A, B, οι οποίοι να δείχνουν πάντοτε σε δυο διαδοχικούς κόμβους. Αρχικά οι A,B, δείχνουν στους κόμβους M_{n+1} και M_{n+2} αντίστοιχα.

Τεχνικές Μείωσης Χώρου

- next(B) = A⊕Link(B)
- previous(A) = Link(A) ⊕ B.



$$\pi.\chi.\ A \oplus \text{Link}(B) = M_4 \oplus M_1 \oplus M_4 = M_1$$

$$\text{Link}(A) \oplus B = M_5 \oplus M_5 \oplus M_3 = M_3$$

C Implementation

From: https://www.geeksforgeeks.org/xor-linked-list-a-memory-efficient-doubly-linked-list-set-2/

```c
/* C/C++ Implementation of Memory
   efficient Doubly Linked List */
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>

// Node structure of a memory
// efficient doubly linked list
struct Node
{
    int data;
    struct Node* npx; /* XOR of next and previous node */
};

/* returns XORed value of the node addresses */
struct Node* XOR (struct Node *a, struct Node *b)
{
    return (struct Node*) ((uintptr_t) (a) ^ (uintptr_t) (b));
}

/* Insert a node at the begining of the
   XORed linked list and makes the newly
   inserted node as head */
void insert(struct Node **head_ref, int data)
{
    // Allocate memory for new node
    struct Node *new_node = (struct Node *) malloc (sizeof (struct Node) );
    new_node->data = data;

    /* Since new node is being inserted at the
       begining, npx of new node will always be
       XOR of current head and NULL */
    new_node->npx = XOR(*head_ref, NULL);

    /* If linked list is not empty, then npx of
       current head node will be XOR of new node
       and node next to current head */
    if (*head_ref != NULL)
    {
        // *(head_ref)->npx is XOR of NULL and next.
        // So if we do XOR of it with NULL, we get next
        struct Node* next = XOR((*head_ref)->npx, NULL);
        (*head_ref)->npx = XOR(new_node, next);
    }

    // Change head
    *head_ref = new_node;
}

// prints contents of doubly linked
// list in forward direction
void printList (struct Node *head)
{
    struct Node *curr = head;
    struct Node *prev = NULL;
    struct Node *next;

    printf ("Following are the nodes of Linked List: \n");

    while (curr != NULL)
    {
        // print current node
        printf ("%d ", curr->data);

        // get address of next node: curr->npx is
        // next^prev, so curr->npx^prev will be
        // next^prev^prev which is next
        next = XOR (prev, curr->npx);

        // update prev and curr for next iteration
        prev = curr;
        curr = next;
    }
}

// Driver program to test above functions
int main ()
{
    /* Create following Doubly Linked List
    head-->40<-->30<-->20<-->10 */
    struct Node *head = NULL;
    insert(&head, 10);
    insert(&head, 20);
    insert(&head, 30);
    insert(&head, 40);

    // print the created list
    printList (head);

    return (0);
}
```