Spring 2019 (Course Number 009952)

Data Structures and Lab
(Lecture 06: XOR and Circular Linked List)

Prof. A. P. Shrestha, Ph.D.
Dept. of Computer Science and Engineering, Sejong University

Last Class

- Doubly linked list
- Doubly linked list operations
  -Insertion, Deletion

Today

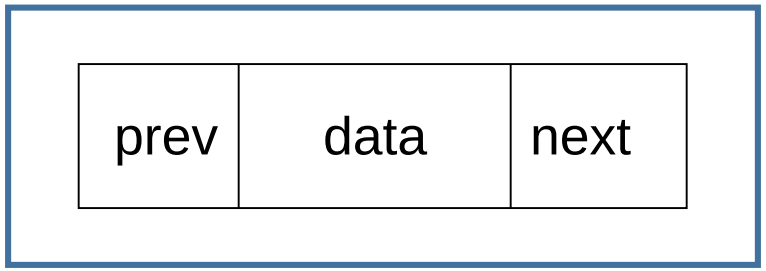
- XOR and Circular linked list
- Josephus Problem

Next class

- Stack

6.1.1 What's Wrong with Doubly Linked List

- It requires more space per node as one extra field is required.
  - Is it possible to implement DLL with single pointer?



- Also, extra procedure is required in all operations as an extra pointer previous should be maintained.

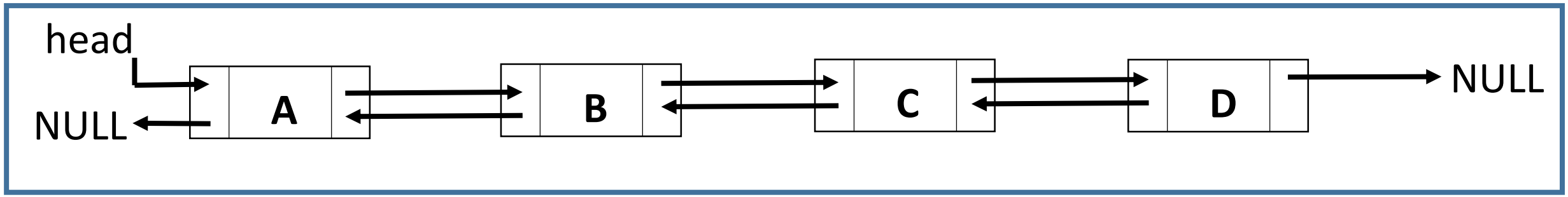
6.1.2 XOR Linked List

- It is also referred as memory efficient doubly linked list
- An XOR list requires only one field with each node other than data field.
- XOR of previous and next addresses is stored

Truth Table for XOR ⊕ Operation

Inputs		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

6.1.3 Idea Behind “From Doubly to XOR Linked List”



Node A

prev=NULL

next=add(B)

npx=0 XOR add(B)

Node B

prev=add(A)

next=add(C)

npx=add(A) XOR add(C)

Node C

prev=add(B)

next=add(D)

npx=add(B) XOR add(D)

Node D

prev=add(C)

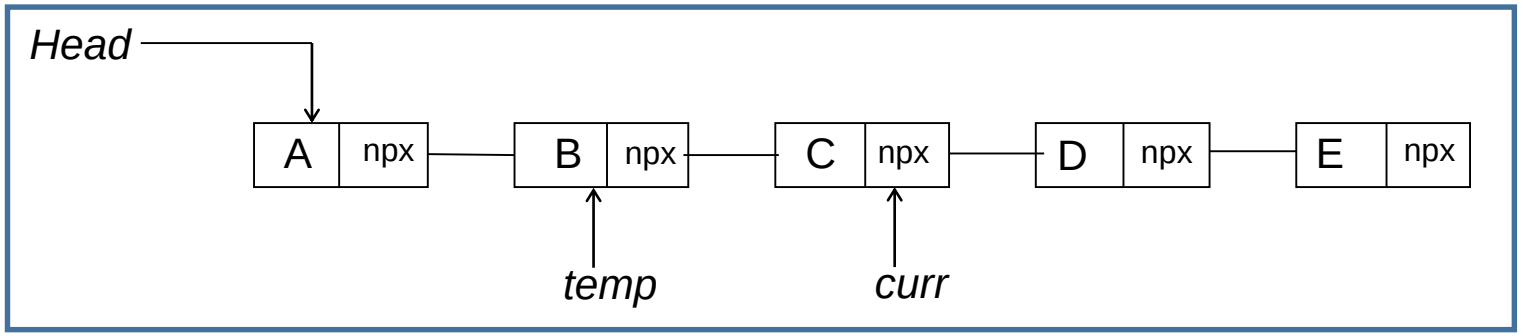
next=NULL

npx= add(C) XOR 0

XOR: bitwise XOR

6.1.4 How XOR List Works?

- We can traverse the XOR list in both forward and reverse direction.
- While traversing the list, we need to remember the address of the previously accessed node in order to calculate the next node's address.



For example:

when we are at node C, we must have address of B.
XOR of add(B) and *npx* of C gives us the add(D).

Forward:

npx(C) is "add(B) XOR add(D)".
Now, npx(C) XOR add(B)
= add(B) XOR add(D) XOR add(B)
= add(D) XOR 0
= add(D)

Backward:

npx(B) is "add(A) XOR add(C)".
Now, npx(B) XOR add(C)
= add(A) XOR add(C) XOR add(C)
= add(A) XOR 0
= add(A)

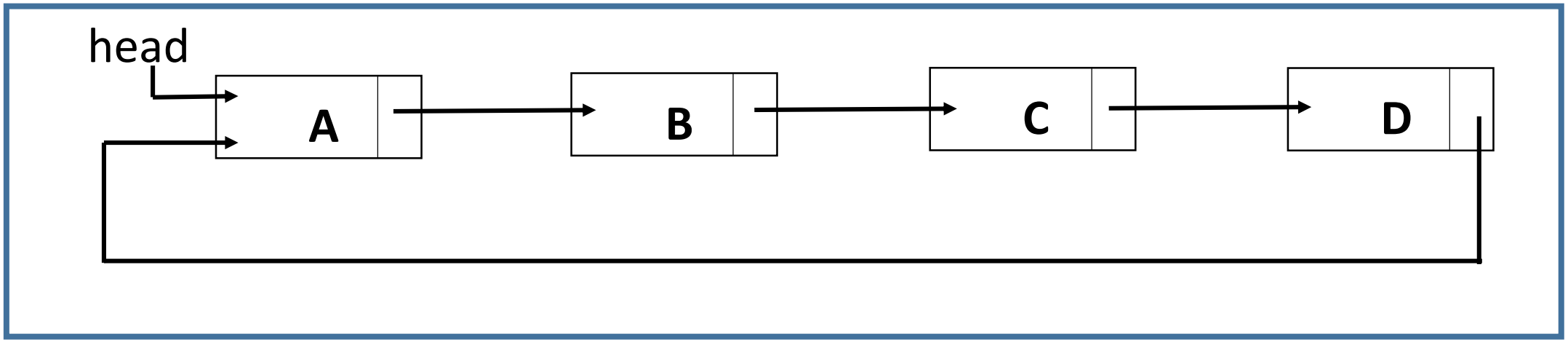
6.1.5 XOR Linked List - Node Implementation

```
struct node
{
        int data;
        struct node *npx;
};
```

```
//XOR the value of two node addresses
struct node* (struct node *a, struct node *b)
        {
                return (struct node *) ((unsigned int) (a)^(unsigined int)(b));
        }
```

6.2.1 Circular Linked List

- All nodes are connected to form a circle.
- A circular linked list can be a singly circular linked list or doubly circular linked list.
- There is no NULL at the end or front.
- While traversing the circular linked lists we should be careful; otherwise we will be traversing the list infinitely



6.2.2 Circular Linked List Applications

- Example 1:
  - Several processes are using the same computer resource (CPU) for the same amount of time
  - Need to assure that no process accesses the resource before all other processes do (round robin algorithm)

    - Time sharing
- Example 2:
  - Keeping track of whose turn it is in a multi-player board game.
  - Put all the players in a circular linked list.
  - After a player takes his turn, advance to the next player in the list.
  - This will cause the program to cycle to repeat among the players (until some termination condition is met)

6.2.3 Circular Linked List: Pros and Cons

Pros

- If we are at a node, then we can go to any node. But, in singly linked list, it is not possible to go to previous node.
- It saves time when we have to go to the first node from the last node. It can be done in single step because there is no need to traverse the in between nodes. But, in double linked list, we have to go through in between nodes.
- The entire list can be traversed starting from any node

Cons

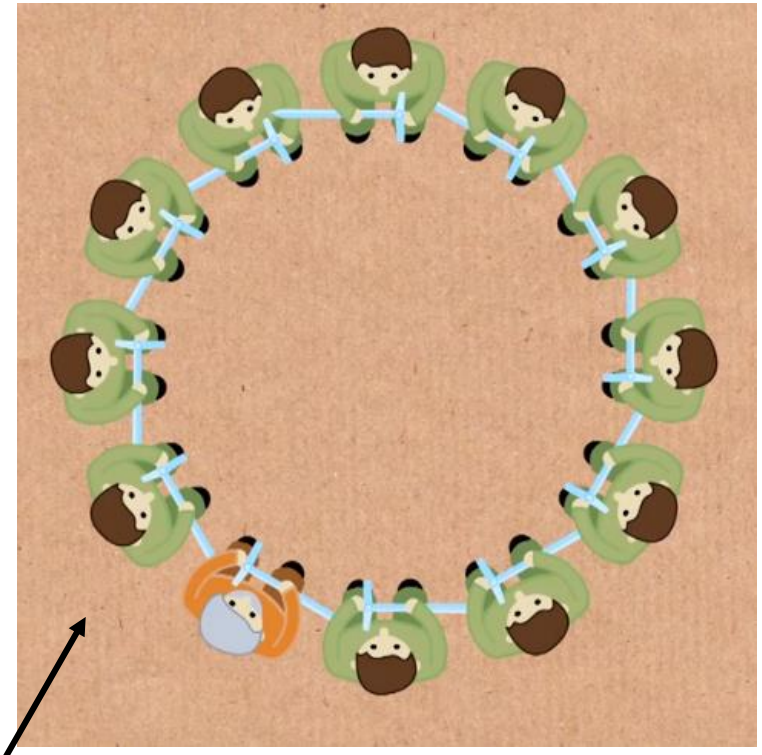
- Finding end of list and loop control is harder (no NULL's to mark beginning and end)
- If proper care is not taken, then the problem of infinite loop can occur.

6.2.4 Circular Linked List - Traversing Implementation

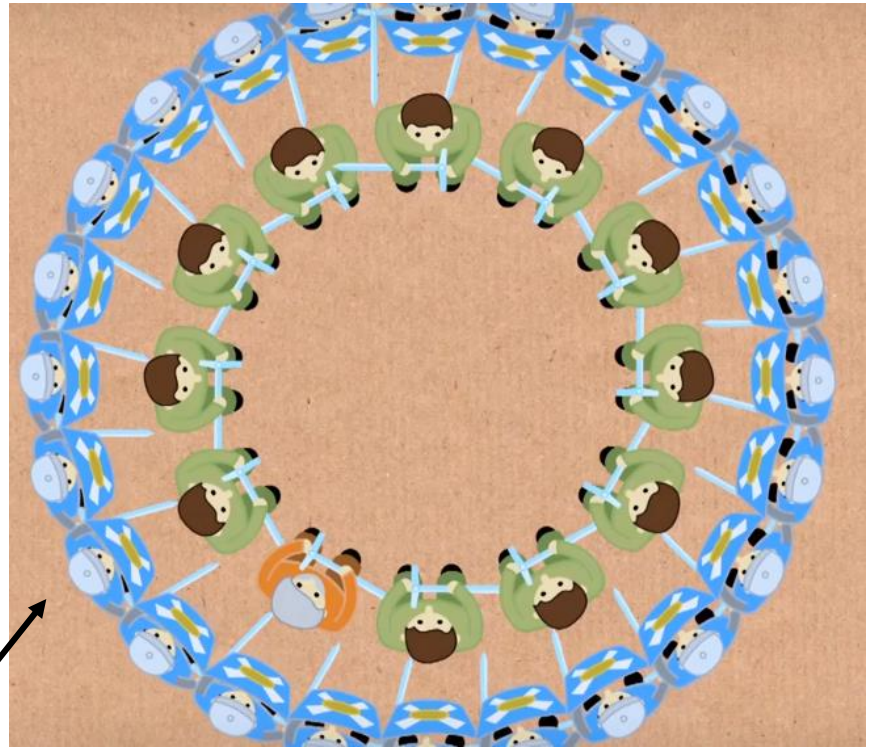
```
void printList(struct node *head)  //no need to pass head if it is accessible
{
        struct node *temp = head;
        if (head != NULL)
        {
                do
                {
                        printf("%d ", temp->data);
                        temp = temp->next;
                } while (temp != head); //note the termination condition
        }
}
```

Q) Is it possible to start traversing from other node than head?

6.3.1 Josephus Problem – Historical Background

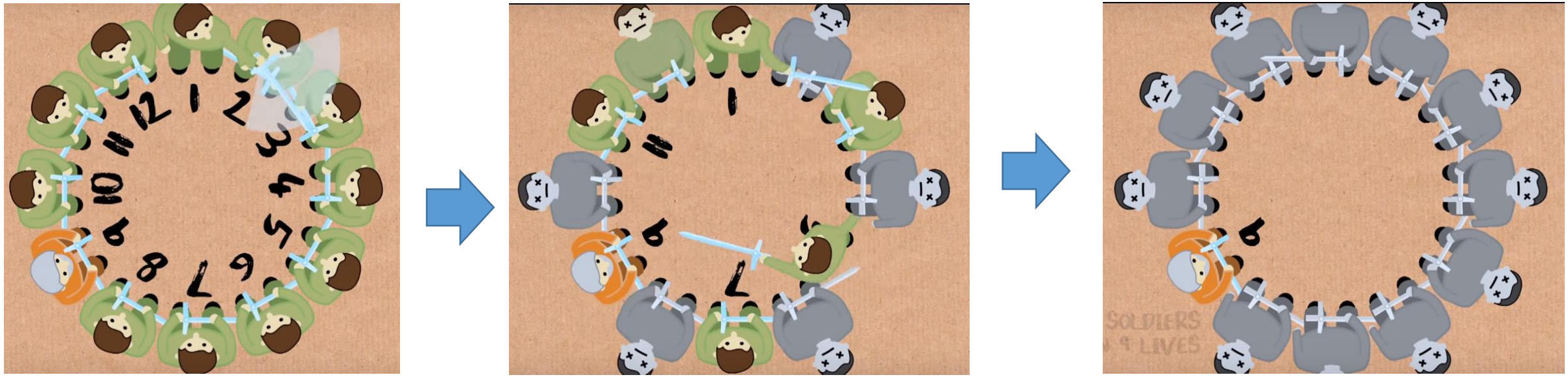


Jewish Army



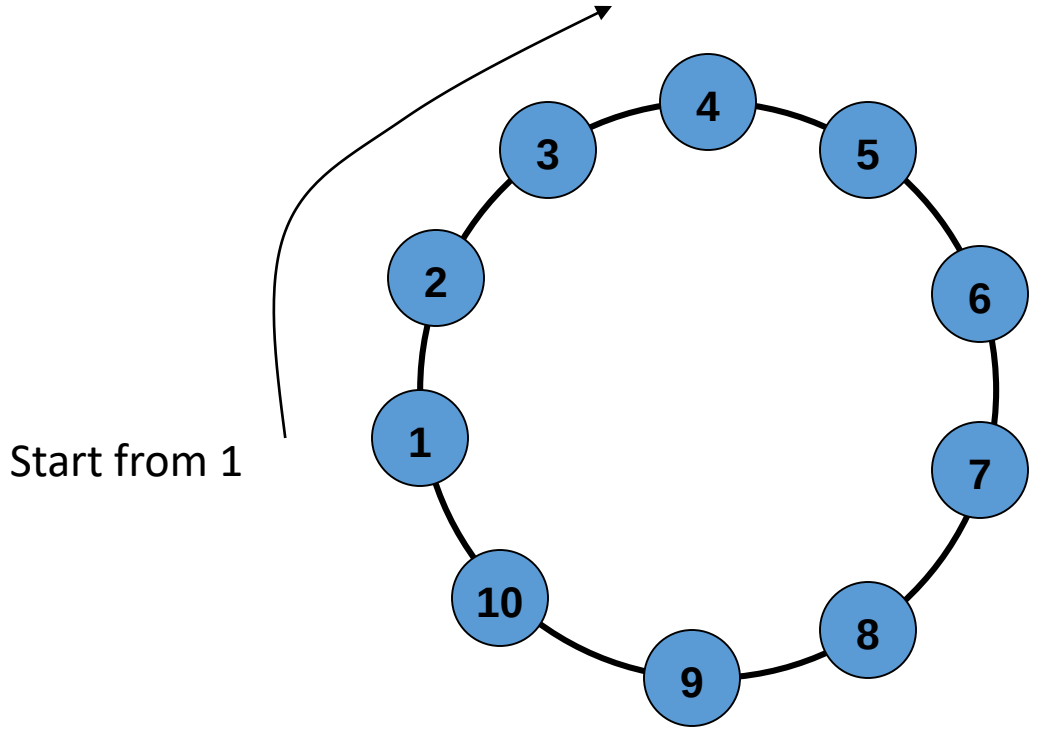
Roman Army

6.3.2 Josephus Problem



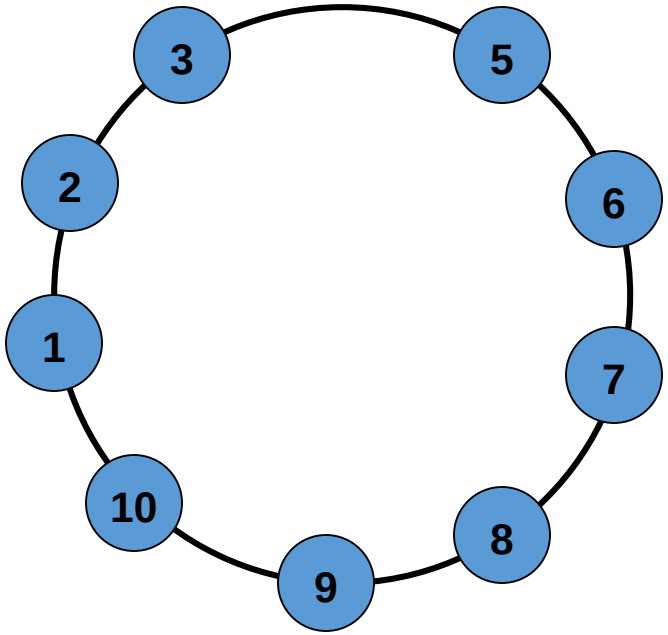
6.3.3 Josephus Problem - Example

- N=10, M=3



6.3.4 Josephus Problem - Example

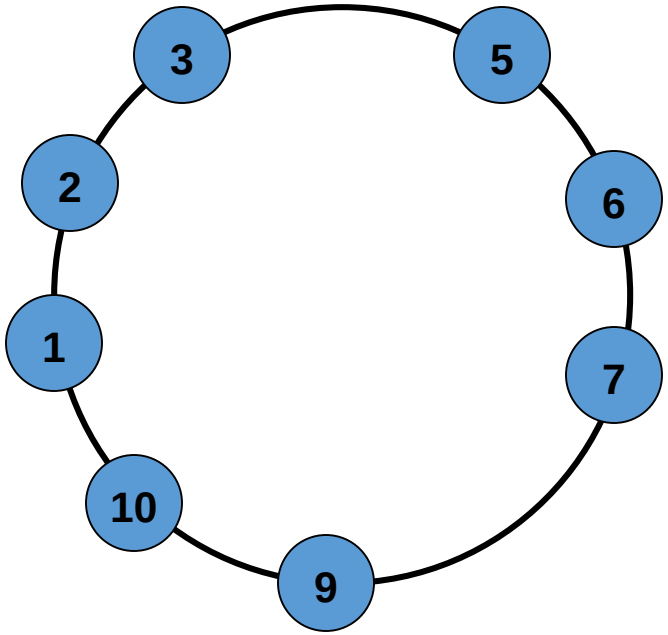
- N=10, M=3



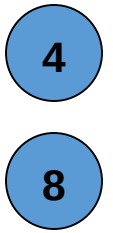
eliminated

6.3.5 Josephus Problem - Example

- N=10, M=3

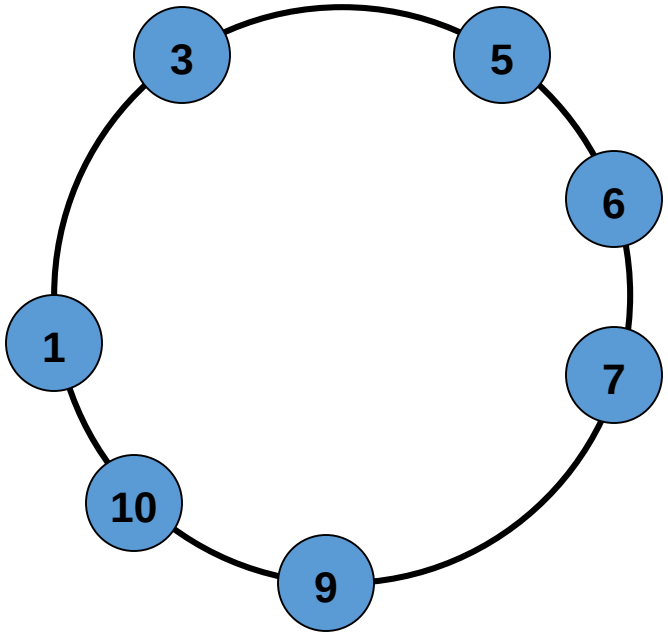


eliminated



6.3.6 Josephus Problem - Example

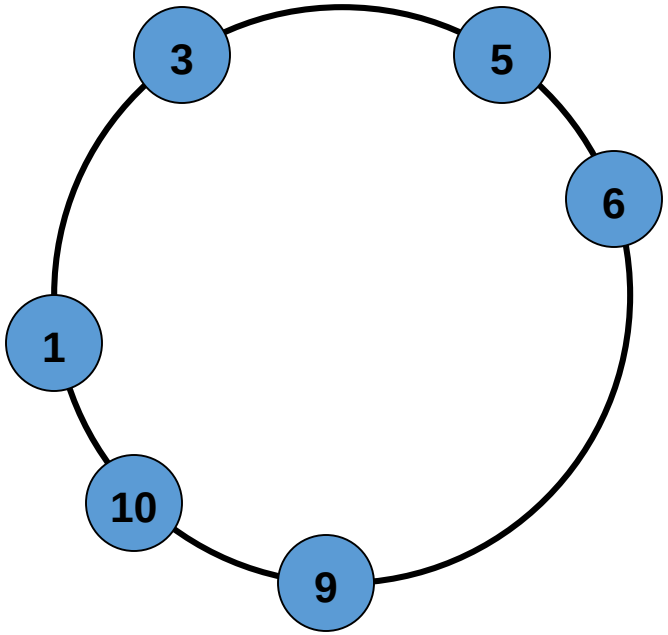
- N=10, M=3



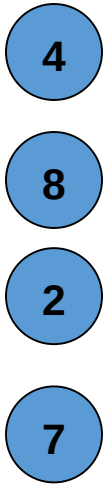
eliminated

6.3.7 Josephus Problem - Example

- N=10, M=3

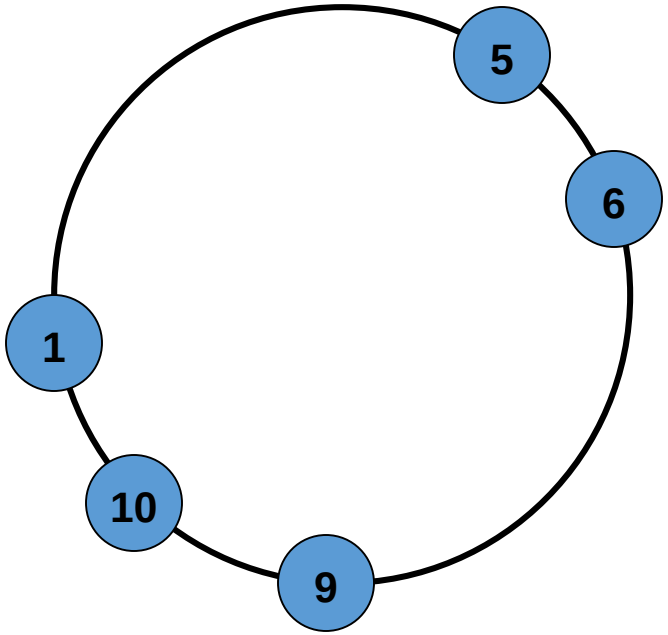


eliminated



6.3.8 Josephus Problem - Example

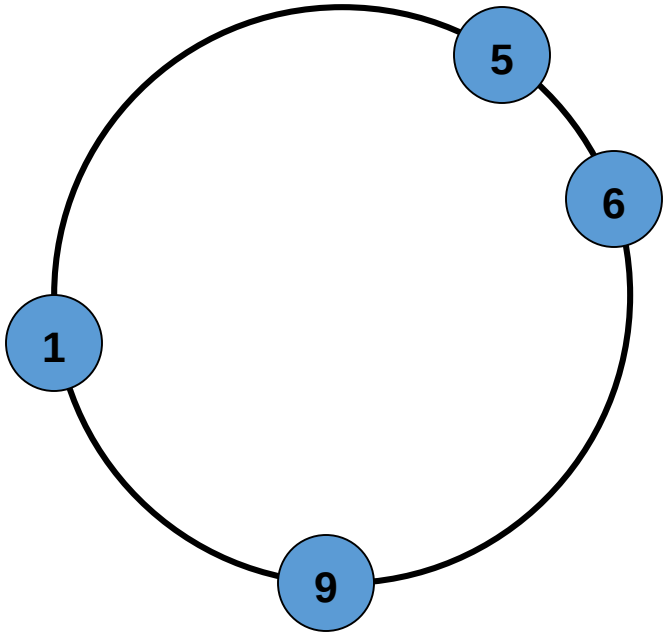
- N=10, M=3



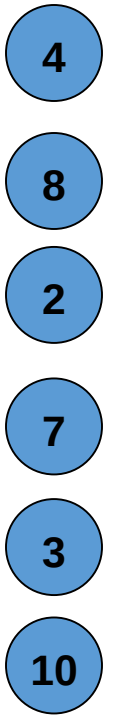
eliminated

6.3.9 Josephus Problem - Example

- N=10, M=3

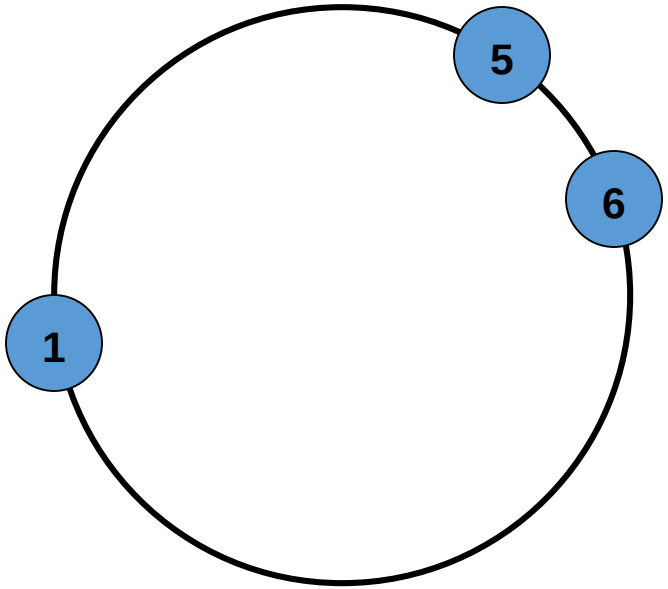


eliminated

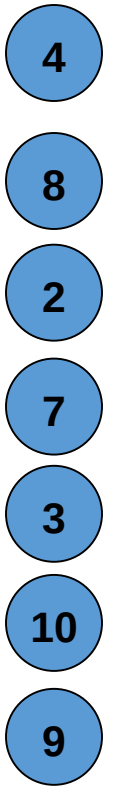


6.3.10 Josephus Problem - Example

- N=10, M=3

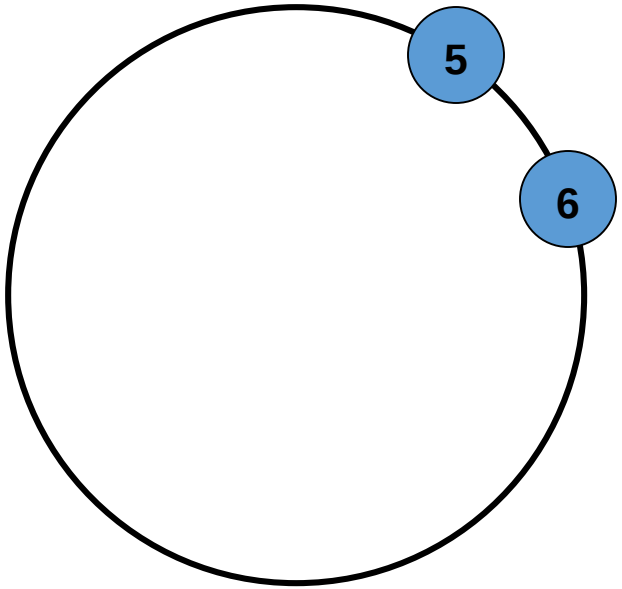


eliminated

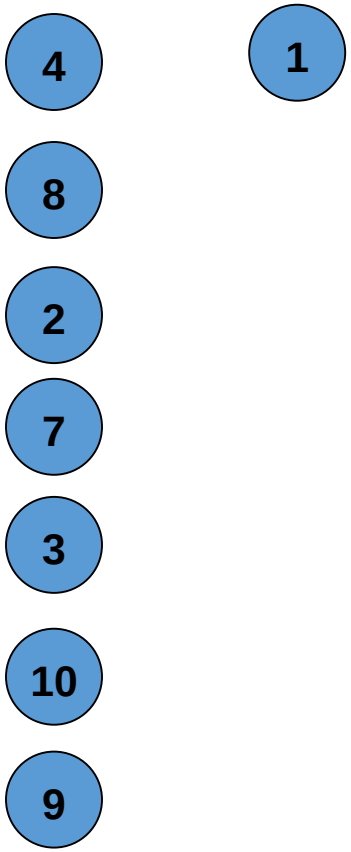


6.3.11 Josephus Problem - Example

- N=10, M=3

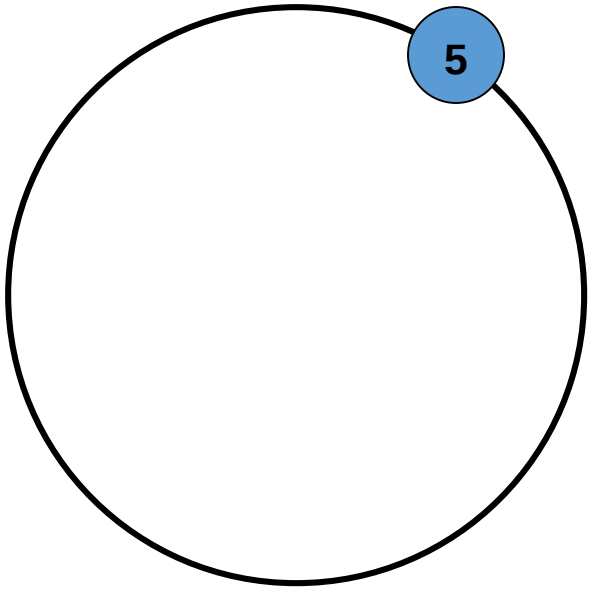


eliminated

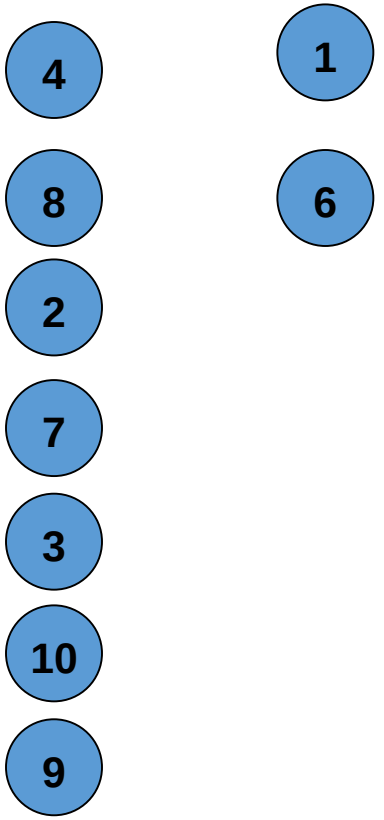


6.3.12 Josephus Problem - Example

- N=10, M=3



eliminated



6.4.1 Josephus Problem - Implementation

```
struct node
{
        int data;
        struct node *next;
};
typedef struct node* nodePtr;
```

```
nodePtr newNode(int data)
{
        nodePtr n = new node;
        n->next = n;
        n->data = data;
}
```

6.4.2 Josephus Problem - Implementation

```
//Total soldiers are n, and skip by m
void josephus(int m, int n)
{
    nodePtr head = newNode(1);
    nodePtr temp = head;
    for (int i = 2; i <= n; i++)
    {
        temp->next = newnode(i);
        temp = temp->next;
     }
     temp->next = head;
    // Connect last node to first
```

```
nodePtr ptr1 = head, ptr2 = head;
while (ptr1->next != ptr1)
{
    int count = 1;
    while (count != m)
    {
        ptr2 = ptr1;
        ptr1 = ptr1->next;
        count++;
    }

    ptr2->next = ptr1->next;
    ptr1  = ptr2->next;
    delete(ptr2);
}

cout << "Last person is \n " << ptr1->data;
} // end of josephus function
```

Q & A ?