Circularly-linked lists

- The next field in the last node in a singly-linked list is set to NULL.
- Moving along a singly-linked list has to be done in a watchful manner.
- Doubly-linked lists have two NULL pointers: prev in the first node and next in the last node.
- A way around this potential hazard is to link the last node with the first node in the list to create a *circularly-linked list*.

Circularly-linked lists

- The next field in the last node in a singly-linked list is set to NULL.
- Moving along a singly-linked list has to be done in a watchful manner.
- Doubly-linked lists have two NULL pointers: prev in the first node and next in the last node.
- A way around this potential hazard is to link the last node with the first node in the list to create a *circularly-linked list*.

Circularly-linked lists

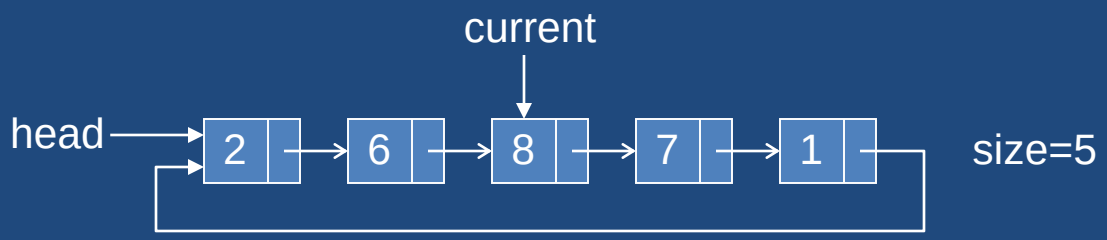
- The next field in the last node in a singly-linked list is set to NULL.
- Moving along a singly-linked list has to be done in a watchful manner.
- Doubly-linked lists have two NULL pointers: prev in the first node and next in the last node.
- A way around this potential hazard is to link the last node with the first node in the list to create a *circularly-linked list*.

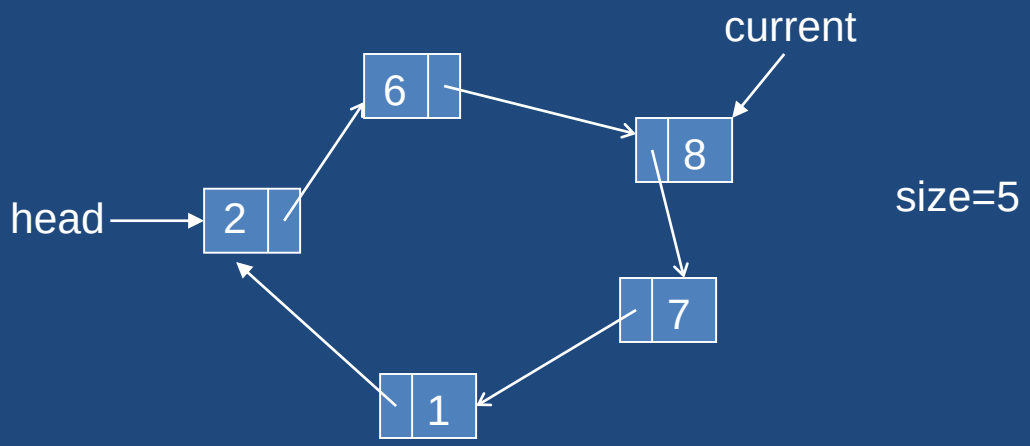
Circularly-linked lists

- The next field in the last node in a singly-linked list is set to NULL.
- Moving along a singly-linked list has to be done in a watchful manner.
- Doubly-linked lists have two NULL pointers: prev in the first node and next in the last node.
- A way around this potential hazard is to link the last node with the first node in the list to create a *circularly-linked list*.

Cicularly Linked List

- Two views of a circularly linked list:





Josephus Problem

- A case where circularly linked list comes in handy is the solution of the *Josephus Problem*.

Josephus Problem

- A case where circularly linked list comes in handy is the solution of the *Josephus Problem*.
- Consider there are 10 persons. They would like to choose a leader.

Josephus Problem

- A case where circularly linked list comes in handy is the solution of the *Josephus Problem*.
- Consider there are 10 persons. They would like to choose a leader.
- The way they decide is that all 10 sit in a circle.

Josephus Problem

- A case where circularly linked list comes in handy is the solution of the *Josephus Problem*.
- Consider there are 10 persons. They would like to choose a leader.
- The way they decide is that all 10 sit in a circle.
- They start a count with person 1 and go in clockwise direction and skip 3. Person 4 reached is eliminated.

Josephus Problem

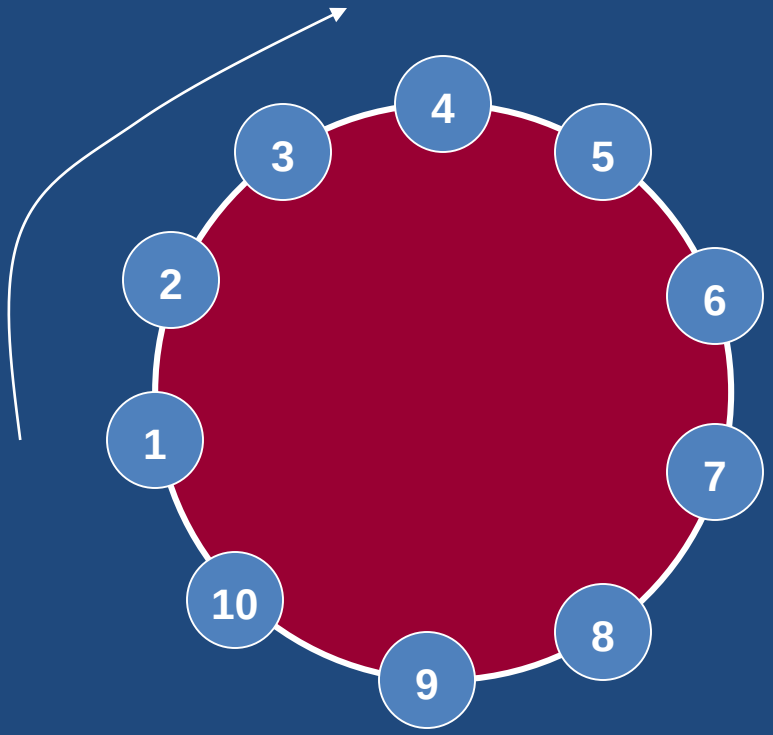
- A case where circularly linked list comes in handy is the solution of the *Josephus Problem*.
- Consider there are 10 persons. They would like to choose a leader.
- The way they decide is that all 10 sit in a circle.
- They start a count with person 1 and go in clockwise direction and skip 3. Person 4 reached is eliminated.
- The count starts with the fifth and the next person to go is the fourth in count.

Josephus Problem

- A case where circularly linked list comes in handy is the solution of the *Josephus Problem*.
- Consider there are 10 persons. They would like to choose a leader.
- The way they decide is that all 10 sit in a circle.
- They start a count with person 1 and go in clockwise direction and skip 3. Person 4 reached is eliminated.
- The count starts with the fifth and the next person to go is the fourth in count.
- Eventually, a single person remains.

Josephus Problem

- N=10, M=3

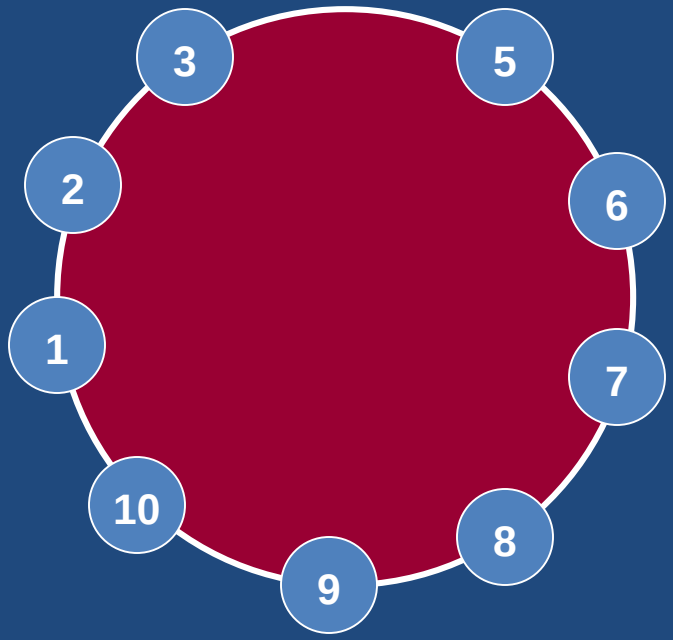


Josephus Problem

- N=10, M=3

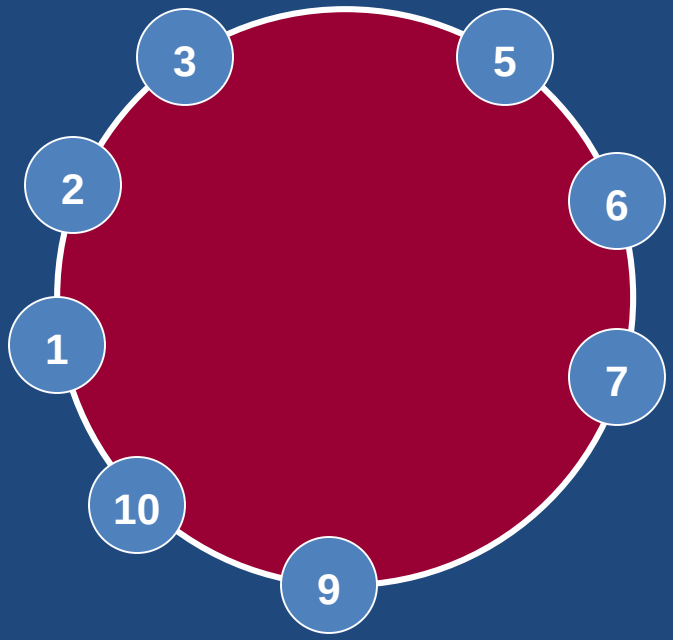
eliminated

4



Josephus Problem

- N=10, M=3



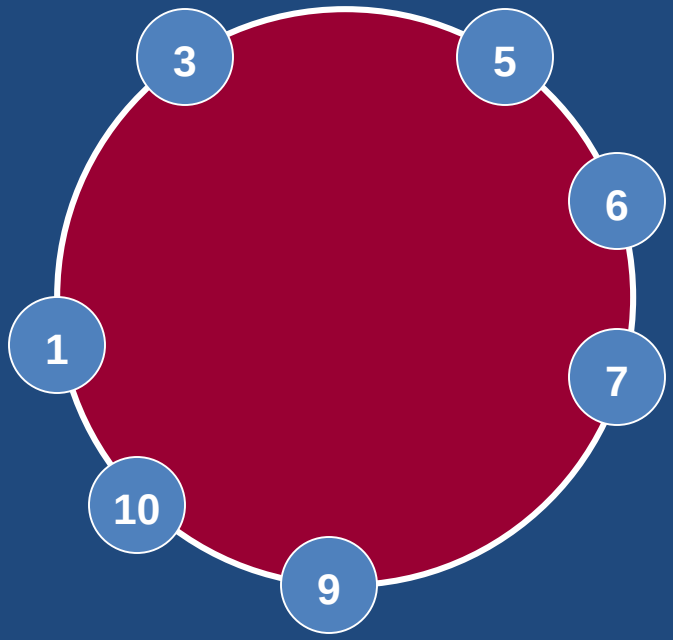
eliminated

4

8

Josephus Problem

- N=10, M=3



eliminated

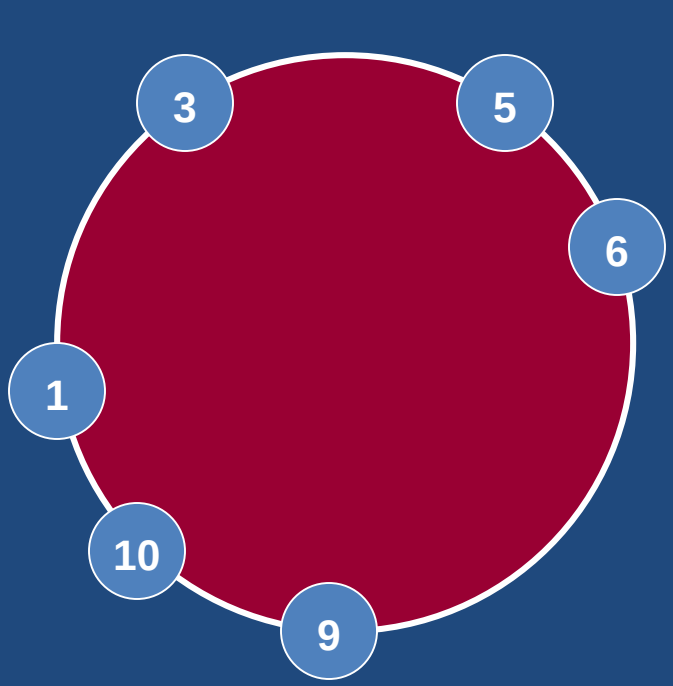
4

8

2

Josephus Problem

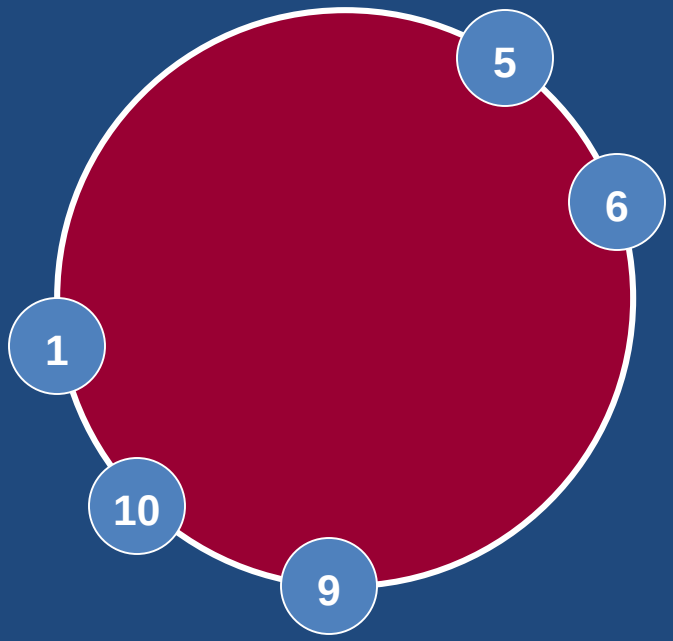
- N=10, M=3



eliminated

Josephus Problem

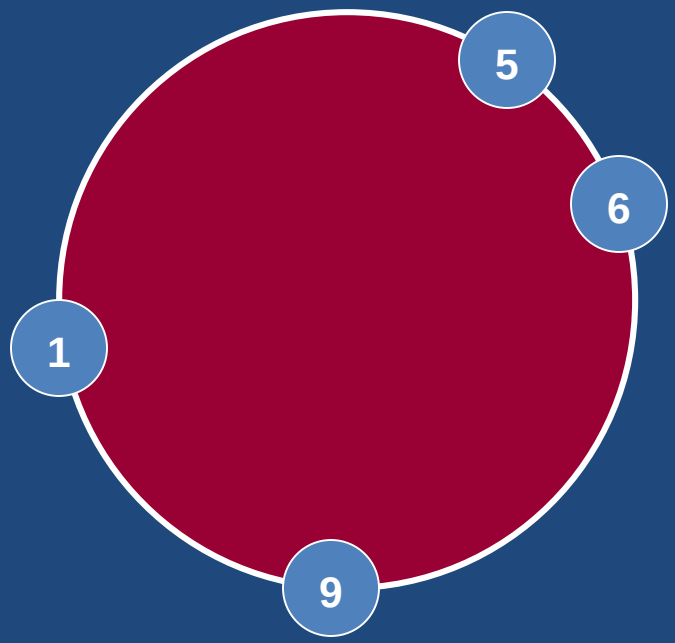
- N=10, M=3



eliminated

Josephus Problem

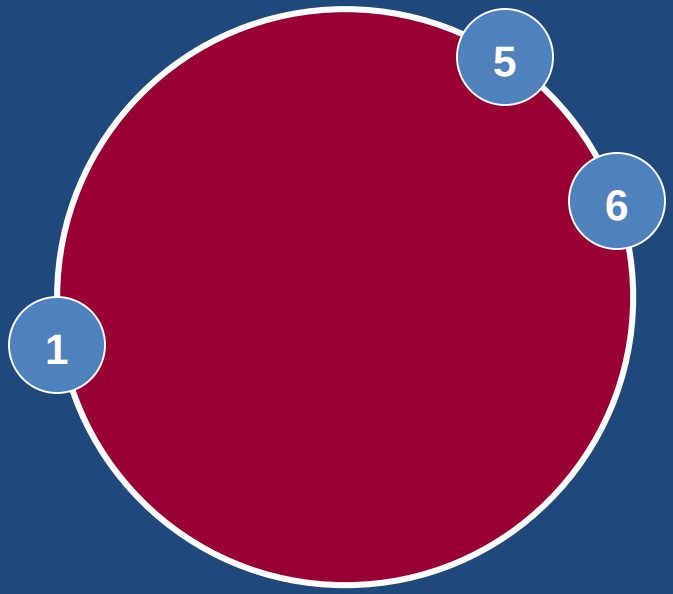
- N=10, M=3



eliminated

Josephus Problem

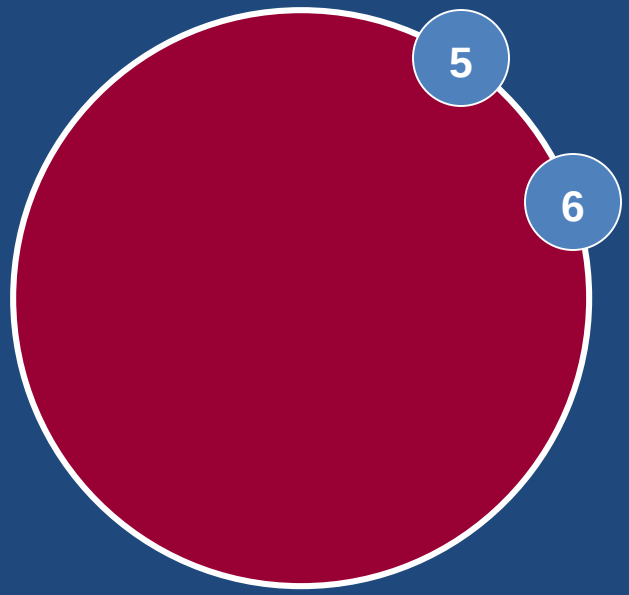
- N=10, M=3



eliminated

Josephus Problem

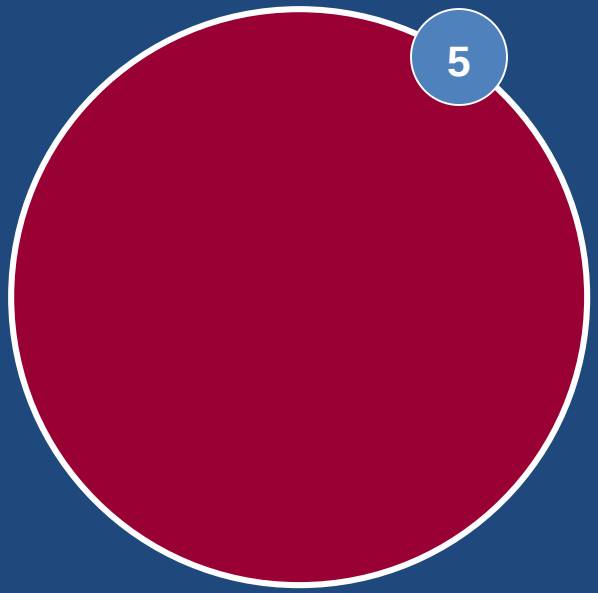
- N=10, M=3



eliminated

Josephus Problem

- N=10, M=3



eliminated

Josephus Problem

```
#include "CList.cpp"
void main(int argc, char *argv[])
{
    CList list;
    int i, N=10, M=3;
    for(i=1; i <= N; i++ ) list.add(i);

    list.start();
    while( list.length() > 1 ) {
        for(i=1; i <= M; i++ ) list.next();
        cout << "remove: " << list.get() << endl;
        list.remove();
    }
    cout << "leader is: " << list.get() << endl;
}
```

Josephus Problem

```
#include "CList.cpp"
void main(int argc, char *argv[])
{
    CList list;
    int i, N=10, M=3;
    for(i=1; i <= N; i++ ) list.add(i);

    list.start();
    while( list.length() > 1 ) {
        for(i=1; i <= M; i++ ) list.next();
        cout << "remove: " << list.get() << endl;
        list.remove();
    }
    cout << "leader is: " << list.get() << endl;
}
```

Josephus Problem

```
#include "CList.cpp"
void main(int argc, char *argv[])
{
    CList list;
    int i, N=10, M=3;
    for(i=1; i <= N; i++ ) list.add(i);

    list.start();
    while( list.length() > 1 ) {
        for(i=1; i <= M; i++ ) list.next();
        cout << "remove: " << list.get() << endl;
        list.remove();
    }
    cout << "leader is: " << list.get() << endl;
}
```

Josephus Problem

```
#include "CList.cpp"
void main(int argc, char *argv[])
{
    CList list;
    int i, N=10, M=3;
    for(i=1; i <= N; i++ ) list.add(i);

    list.start();
    while( list.length() > 1 ) {
        for(i=1; i <= M; i++ ) list.next();
        cout << "remove: " << list.get() << endl;
        list.remove();
    }
    cout << "leader is: " << list.get() << endl;
}
```

Josephus Problem

```
#include "CList.cpp"
void main(int argc, char *argv[])
{
    CList list;
    int i, N=10, M=3;
    for(i=1; i <= N; i++ ) list.add(i);

    list.start();
    while( list.length() > 1 ) {
        for(i=1; i <= M; i++ ) list.next();
        cout << "remove: " << list.get() << endl;
        list.remove();
    }
    cout << "leader is: " << list.get() << endl;
}
```

Josephus Problem

```
#include "CList.cpp"
void main(int argc, char *argv[])
{
    CList list;
    int i, N=10, M=3;
    for(i=1; i <= N; i++ ) list.add(i);

    list.start();
    while( list.length() > 1 ) {
        for(i=1; i <= M; i++ ) list.next();
        cout << "remove: " << list.get() << endl;
        list.remove();
    }
    cout << "leader is: " << list.get() << endl;
}
```

Josephus Problem

```
#include "CList.cpp"
void main(int argc, char *argv[])
{
    CList list;
    int i, N=10, M=3;
    for(i=1; i <= N; i++ ) list.add(i);

    list.start();
    while( list.length() > 1 ) {
        for(i=1; i <= M; i++ ) list.next();
        cout << "remove: " << list.get() << endl;
        list.remove();
    }
    cout << "leader is: " << list.get() << endl;
}
```

Josephus Problem

```
#include "CList.cpp"
void main(int argc, char *argv[])
{
    CList list;
    int i, N=10, M=3;
    for(i=1; i <= N; i++ ) list.add(i);

    list.start();
    while( list.length() > 1 ) {
        for(i=1; i <= M; i++ ) list.next();
        cout << "remove: " << list.get() << endl;
        list.remove();
    }
    cout << "leader is: " << list.get() << endl;
}
```

Josephus Problem

```
#include "CList.cpp"
void main(int argc, char *argv[])
{
    CList list;
    int i, N=10, M=3;
    for(i=1; i <= N; i++ ) list.add(i);

    list.start();
    while( list.length() > 1 ) {
        for(i=1; i <= M; i++ ) list.next();
        cout << "remove: " << list.get() << endl;
        list.remove();
    }
    cout << "leader is: " << list.get() << endl;
}
```

Josephus Problem

```
// Add elements in the list
   int i, N=10, M=3;
   for(i=1; i <= N; i++ ) list.add(i);

//Start josephus selection

   list.start();
   while( list.length() > 1 )
       for(i=1; i <= M; i++ ) list.next();
       cout << "remove: " << list.get() << endl;
       list.remove();

   cout << "leader is: " << list.get() << endl;
```

Josephus Problem

- Using a circularly-linked list made the solution trivial.

Josephus Problem

- Using a circularly-linked list made the solution trivial.
- The solution would have been more difficult if an array had been used.

Josephus Problem

- Using a circularly-linked list made the solution trivial.
- The solution would have been more difficult if an array had been used.
- This illustrates the fact that the choice of the appropriate data structures can significantly simplify an algorithm. It can make the algorithm much faster and efficient.