LECTURE 12

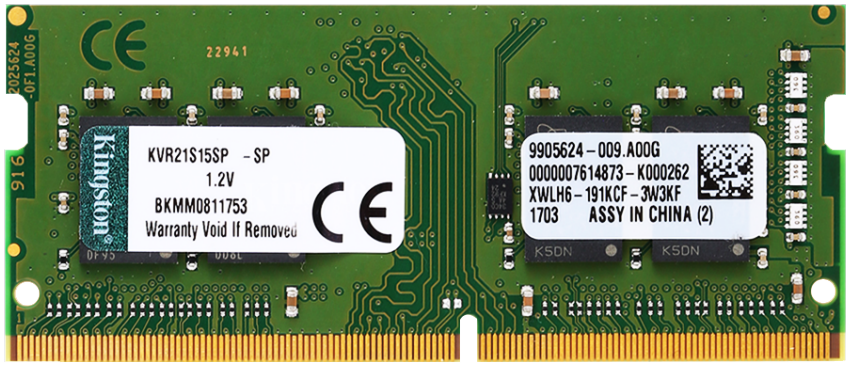
LINKED LIST

Big-O Coding

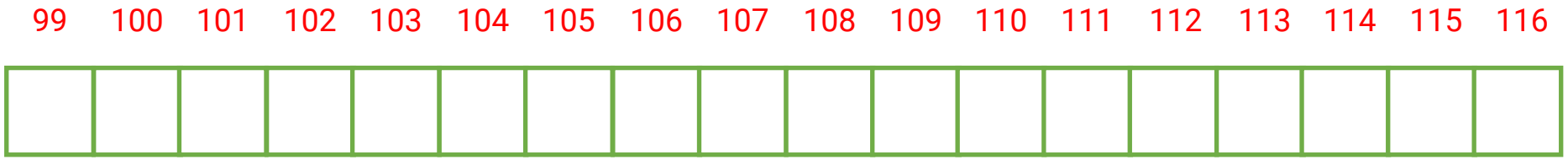
Website: www.bigocoding.com

RAM

RAM (Random Access Memory)

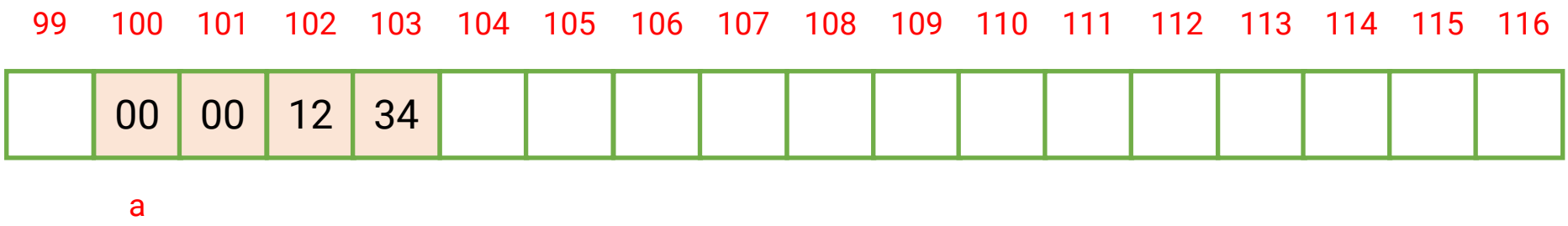


RAM (Random Access Memory)



Biến

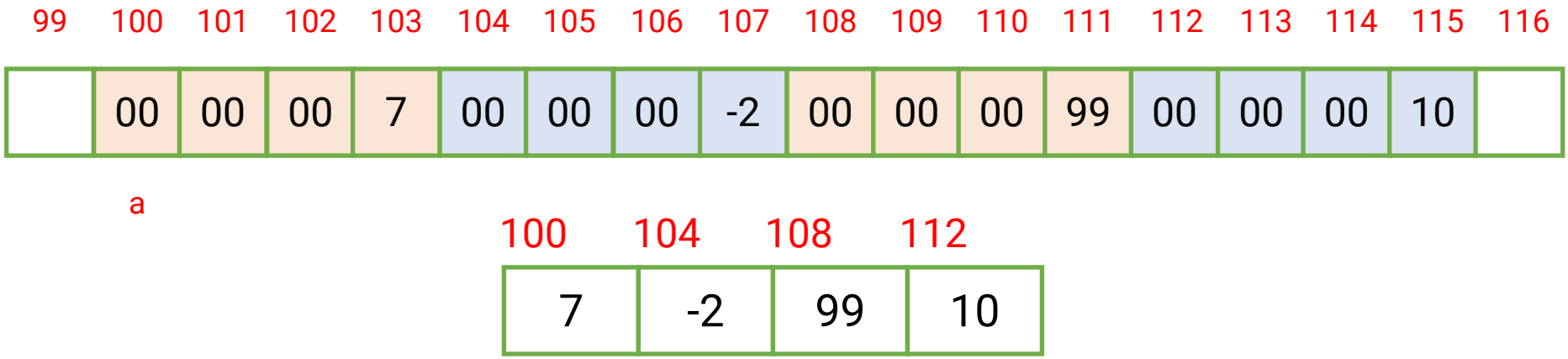
- Khi ta khai báo một biến trong source code, một vùng nhớ (memory block) trong bộ nhớ sẽ được cấp phát để lưu giá trị của biến đó.



Array

Array

- Khi khai báo một array trong source code, một **vùng nhớ liên tiếp** trong bộ nhớ sẽ được cấp phát để lưu các giá trị các phần tử trong array đó.



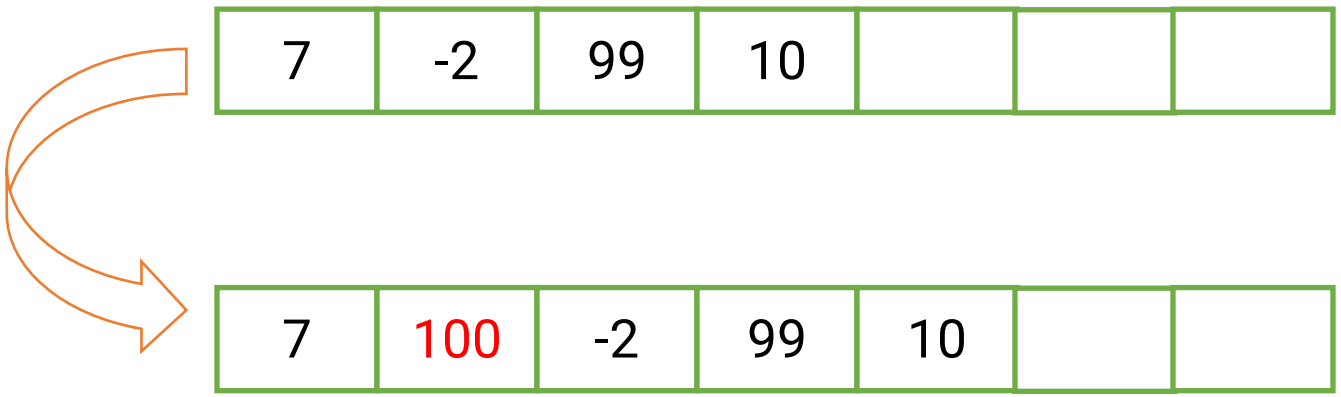
Quiz

- Có mảng a, 1.000.000 phần tử
- Địa chỉ của a[0]: 100
- Địa chỉ của a[6789]: ?

Điểm mạnh của array

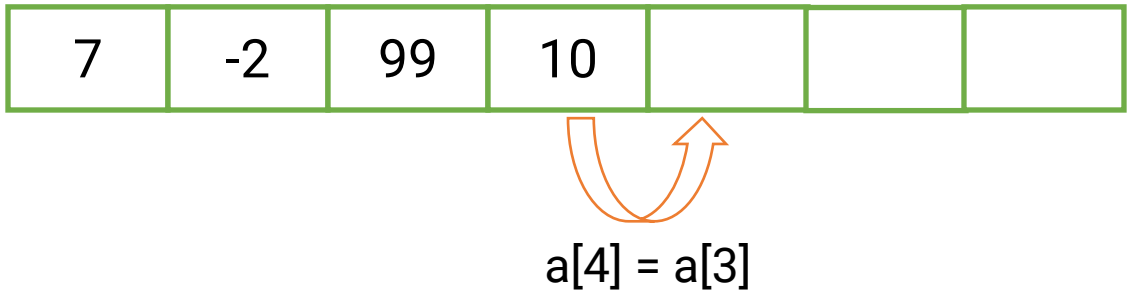
- Độ phức tạp thời gian của thao tác lấy / cập nhật giá trị phần tử thứ i trong mảng: **O(1)**.

insert(a, 1, 100)



insert(a, 1, 100) – Thực hiện (C++)

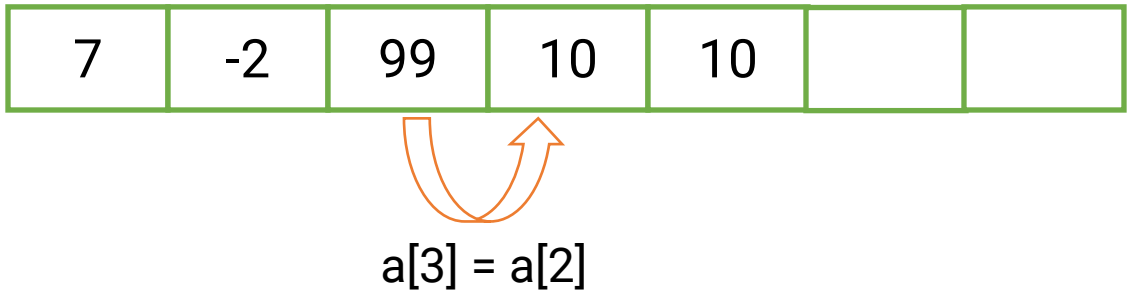
- Dời chỗ



insert(a, 1, 100) – Thực hiện (C++)

insert(a, 1, 100) – Thực hiện (C++)

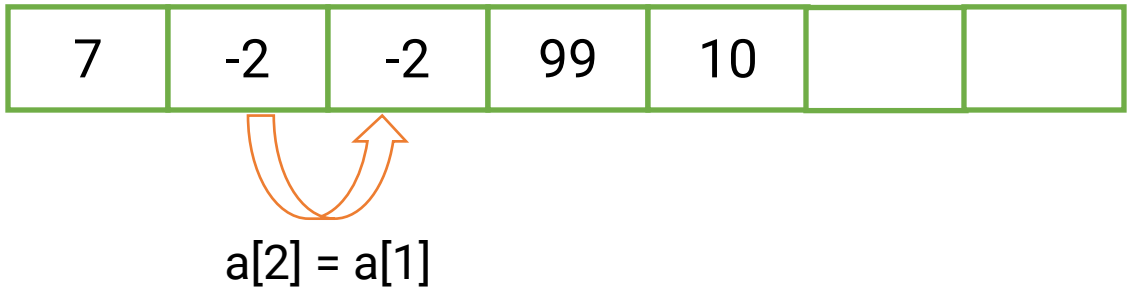
- Dời chỗ



insert(a, 1, 100) – Thực hiện (C++)

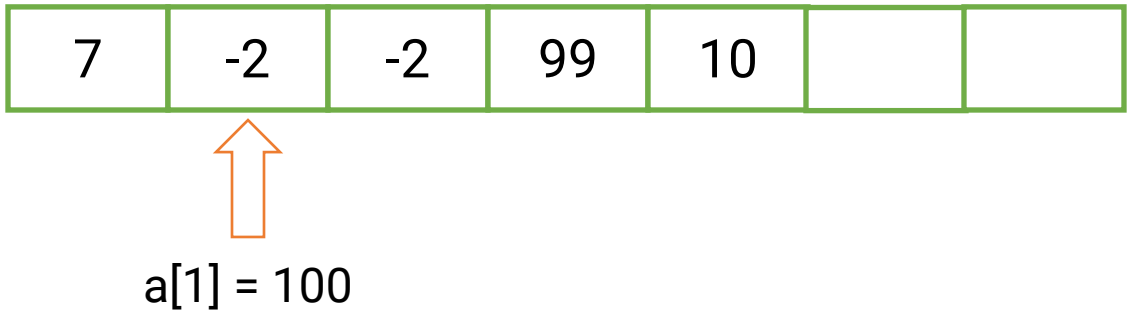
insert(a, 1, 100) – Thực hiện (C++)

- Dời chỗ



insert(a, 1, 100) – Thực hiện (C++)

- Đưa 100 vào vị trí 1



insert(a, 1, 100) – Thực hiện (C++)

insert(a, k, x) – Source code (C++)

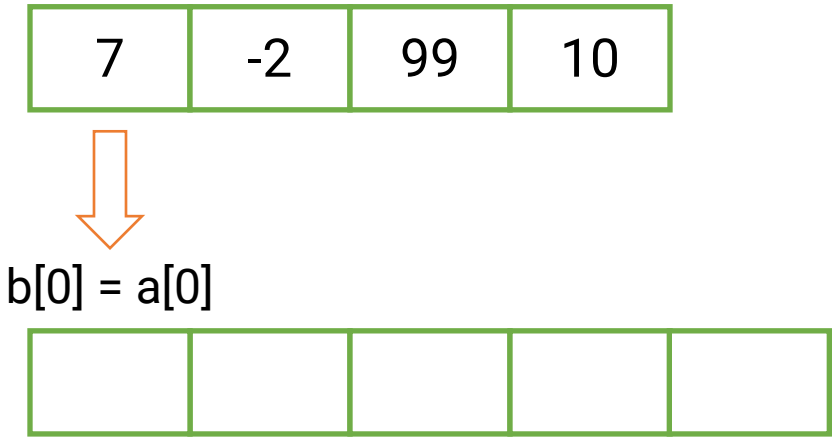
```
int a[100] = {7, -2, 99, 10};
int n = 4;

int x = 100;
int k = 1;

if(k >= 0 && k <= n){
    for(int i = n; i > k; i--)
        a[i] = a[i - 1];

    a[k] = x;

    n++;
}
```

insert(a, 1, 100) – Thực hiện (Java)

- Tạo mảng b, có kích thước n + 1

insert(a, 1, 100) – Thực hiện (Java)

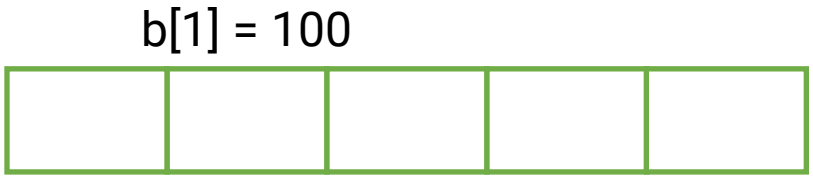
- Sao chép mảng a qua mảng b



insert(a, 1, 100) – Thực hiện (Java)

insert(a, 1, 100) – Thực hiện (Java)

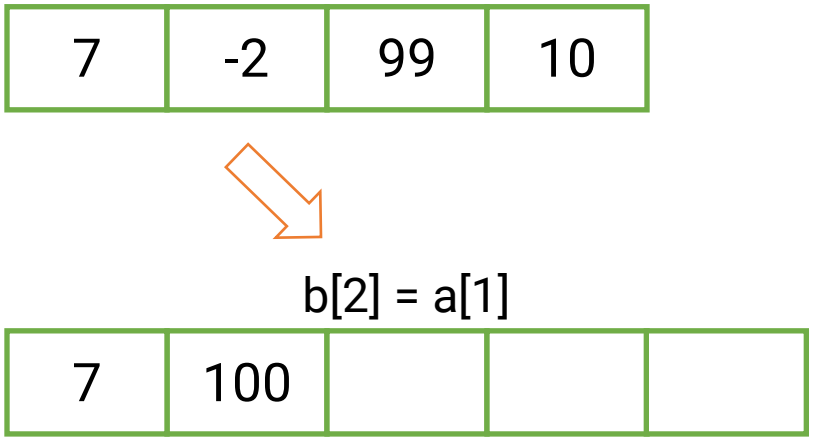
- Đưa 100 vào vị trí a[1]



insert(a, 1, 100) – Thực hiện (Java)

insert(a, 1, 100) – Thực hiện (Java)

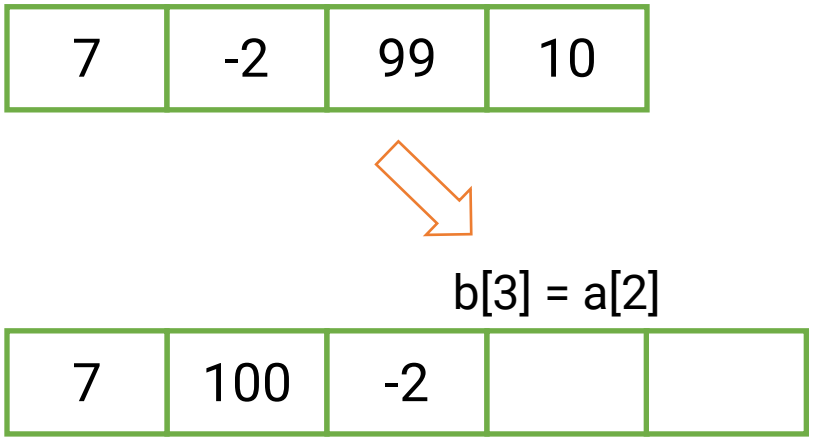
- Tiếp tục sao chép mảng a qua mảng b



insert(a, 1, 100) – Thực hiện (Java)

insert(a, 1, 100) – Thực hiện (Java)

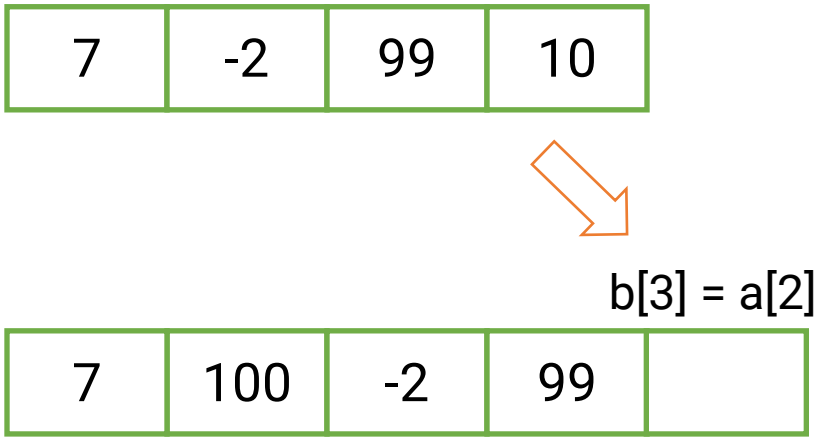
- Tiếp tục sao chép mảng a qua mảng b



insert(a, 1, 100) – Thực hiện (Java)

insert(a, 1, 100) – Thực hiện (Java)

- Tiếp tục sao chép mảng a qua mảng b



insert(a, 1, 100) – Thực hiện (Java)

insert(a, k, x) (Java)

```java
int []a = {7, -2, 99, 10};
int []b;
int x = 100;
int k = 1;

if(k >= 0 && k <= a.length){
    b = new int[a.length + 1];
    for(int i = a.length; i > k; i--){
        b[i] = a[i-1];
    }

    b[k] = x;

    for(int i = k - 1; i >= 0; i--){
        b[i] = a[i];
    }

    a = b;
}
```

insert(a, k, x) (Python)

- Array trong Python đã có sẵn hàm insert, hoạt động tương tự với ví dụ trên.

```
a = [7, -2, 99, 10]
x = 100
k = 1

# Hàm insert() hoạt động tương tự
if(k >= 0 and k <= len(a)):
    a.insert(1, 100)
```

Quiz (C++)

- Giả sử, khai báo a tối đa 1.000.000 phần tử, n hiện tại là 500.000 phần tử.
- Cần insert x = 7 vào vị trí k = 499.500.
- Cần thực hiện bao nhiêu thao tác gán?

Quiz (C++)

- Giả sử, khai báo a tối đa 1.000.000 phần tử, n hiện tại là 1.000.000 phần tử.
- Chuyện gì xảy ra nếu ta muốn insert x = 8 vào cuối mảng?

Quiz (Java)

- Giả sử, a hiện tại có 500.000 phần tử.
- Cần insert x = 7 vào vị trí k = 499.500.
- Cần thực hiện bao nhiêu thao tác gán?

Điểm yếu của array

- Độ phức tạp thời gian của thao tác insert(a, k, x): **O(n)**.

- Tương tự, độ phức tạp thời gian của thao tác remove(a, k) : **O(n)**.
  - Xem các slide sau.

remove(a, k) (C++)

```cpp
int a[100] = {7, 100, -2, 99, 10};
int n = 5;

int k = 2;

if(k >= 0 && k <= n - 1){
    for(int i = k; i < n - 1; i++)
        a[i] = a[i + 1];
    n--;
}
```

remove(a, k) (Java)

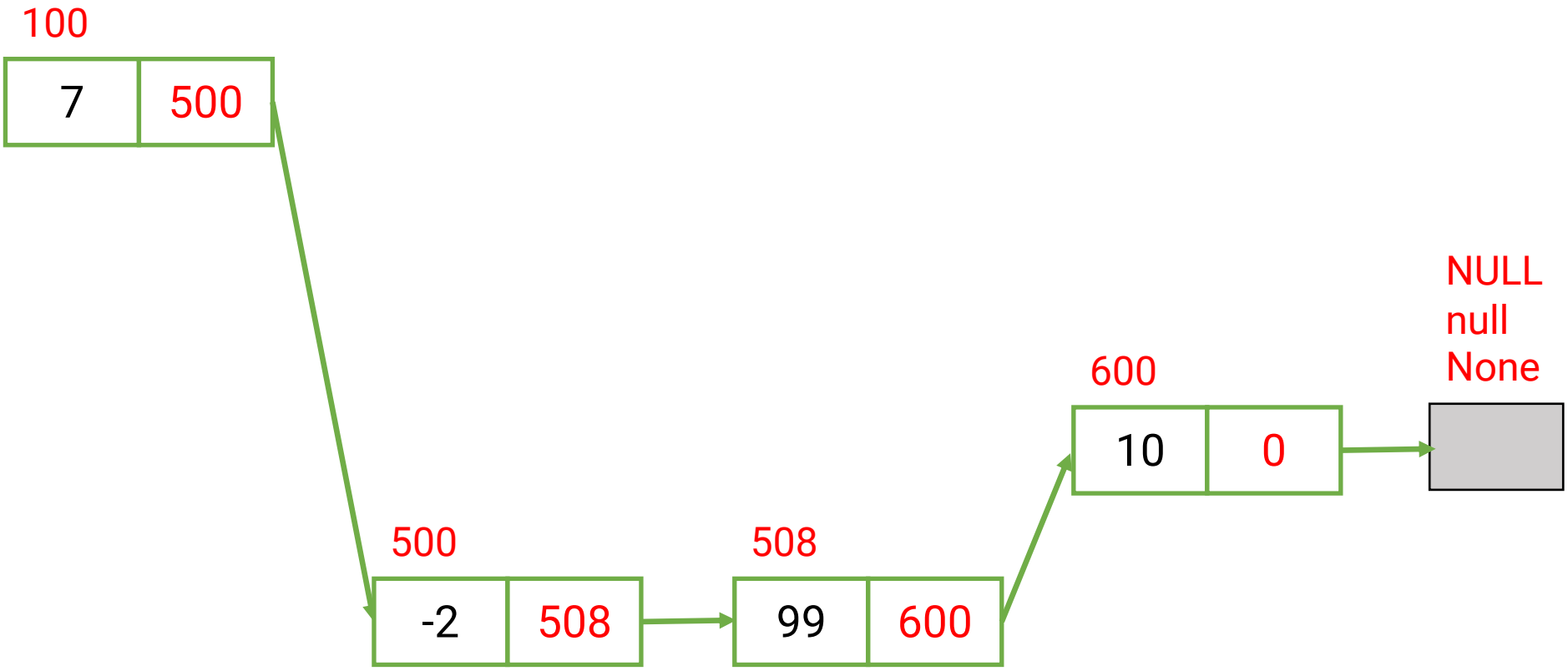
```java
int []a = {7, 100, -2, 99, 10};
int []b;

int k = 2;

if(k >= 0 && k <= a.length - 1){
    b = new int[a.length - 1];
    for(int i = k; i < a.length - 1; i++){
        b[i] = a[i + 1];
    }


    for(int i = 0; i <= k - 1; i++){
        b[i] = a[i];
    }

    a = b;
}
```

remove(a, k) (Python)

```
a = [7, 100, -2, 99, 10]
k = 2

# Hàm pop() hoạt động tương tự
if(k >= 0 and k <= len(a) - 1):
    a.pop(k)
```

Linked list

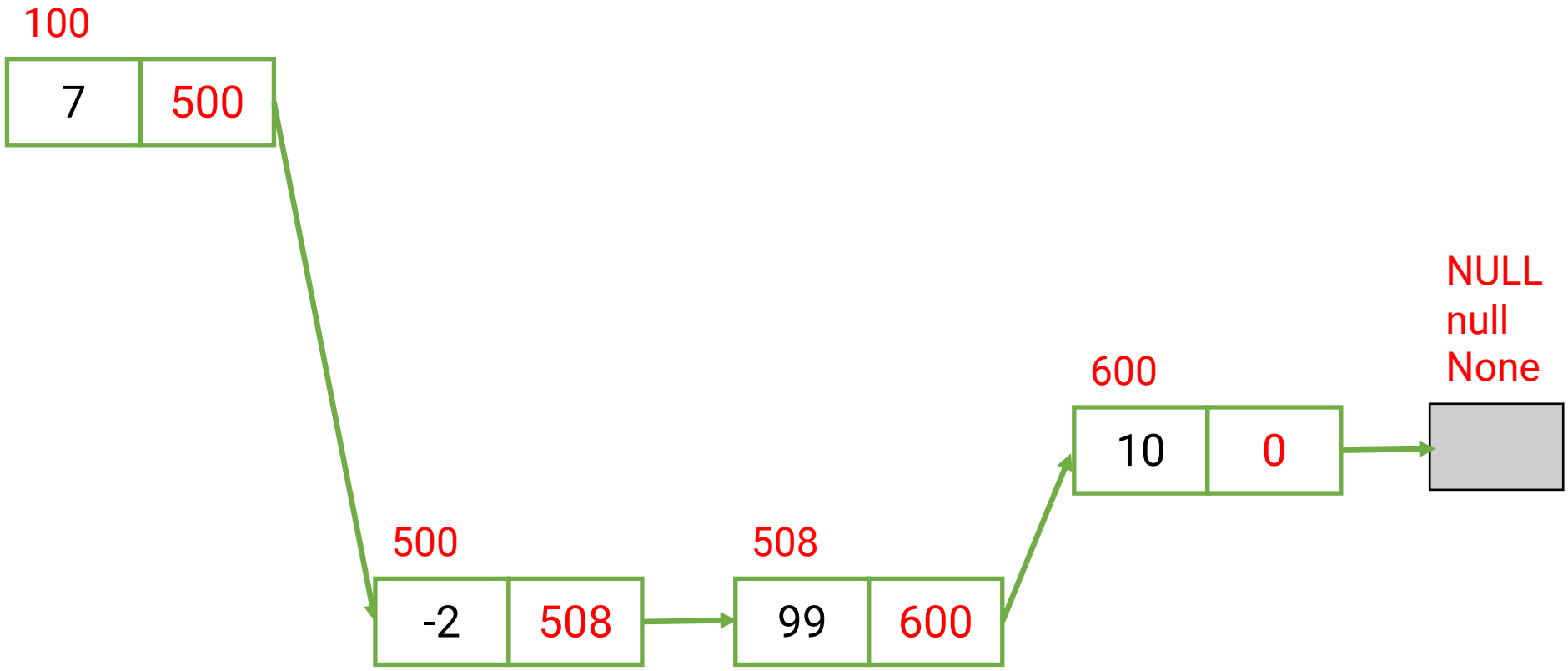
Linked list



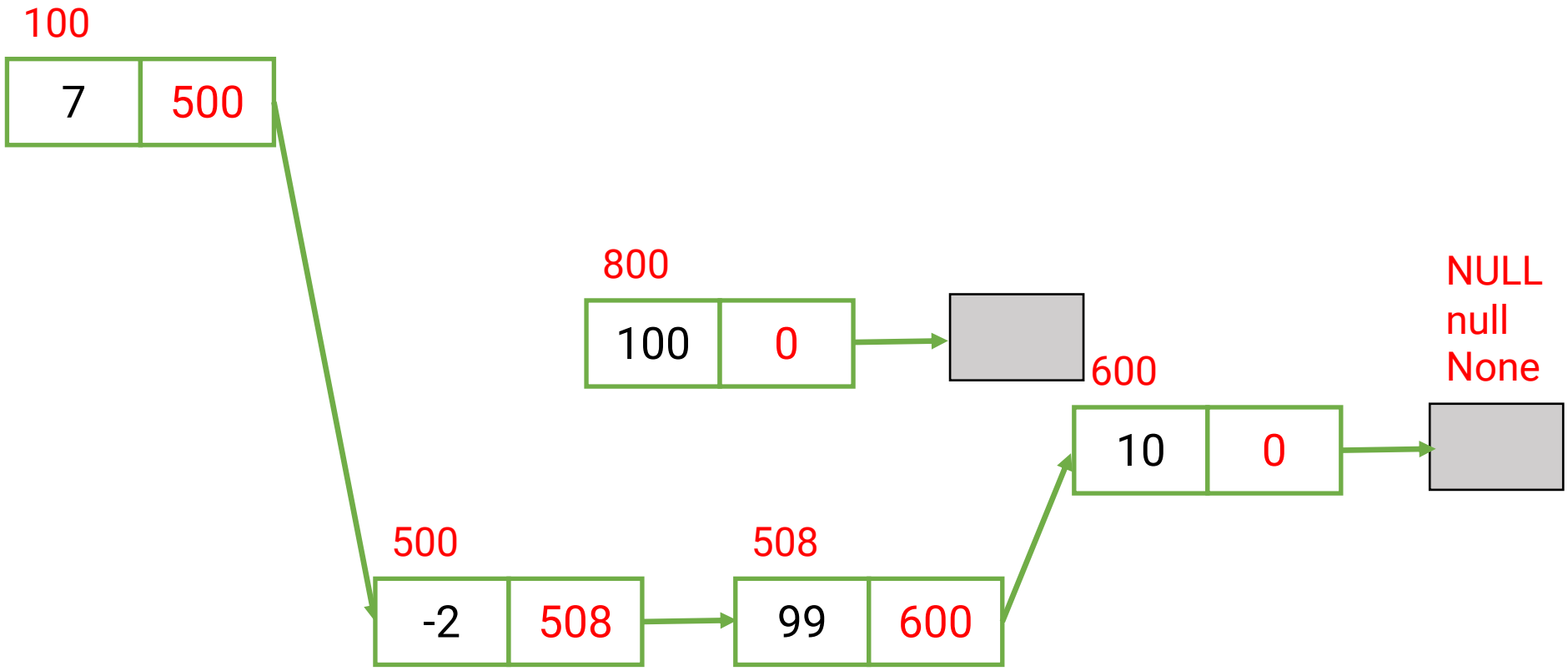
Liên hệ thực tế

- 5 người bạn rủ nhau đi xem film.
- TH1: Phim hot, đặt vé trễ.
- TH2: Phim ít người coi.

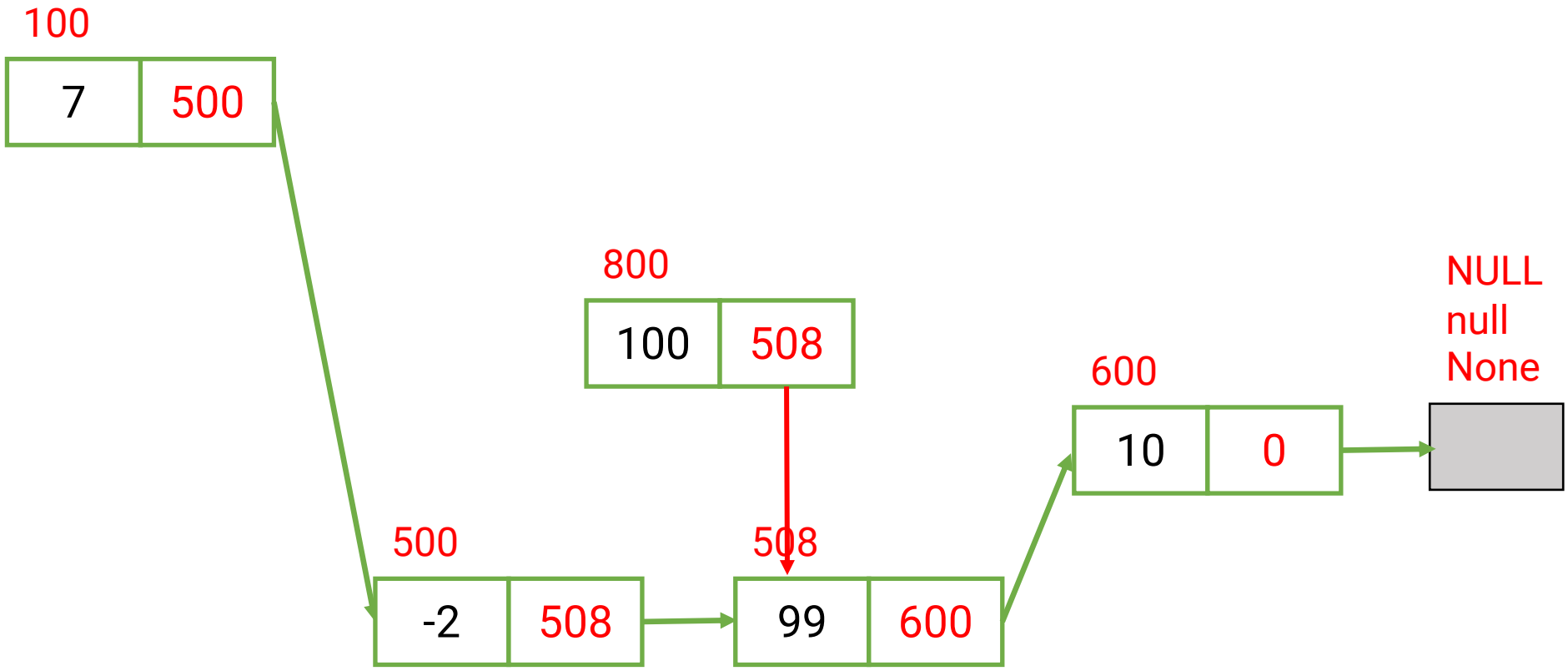
insertAfter(lst, -2, 100) – Thực hiện



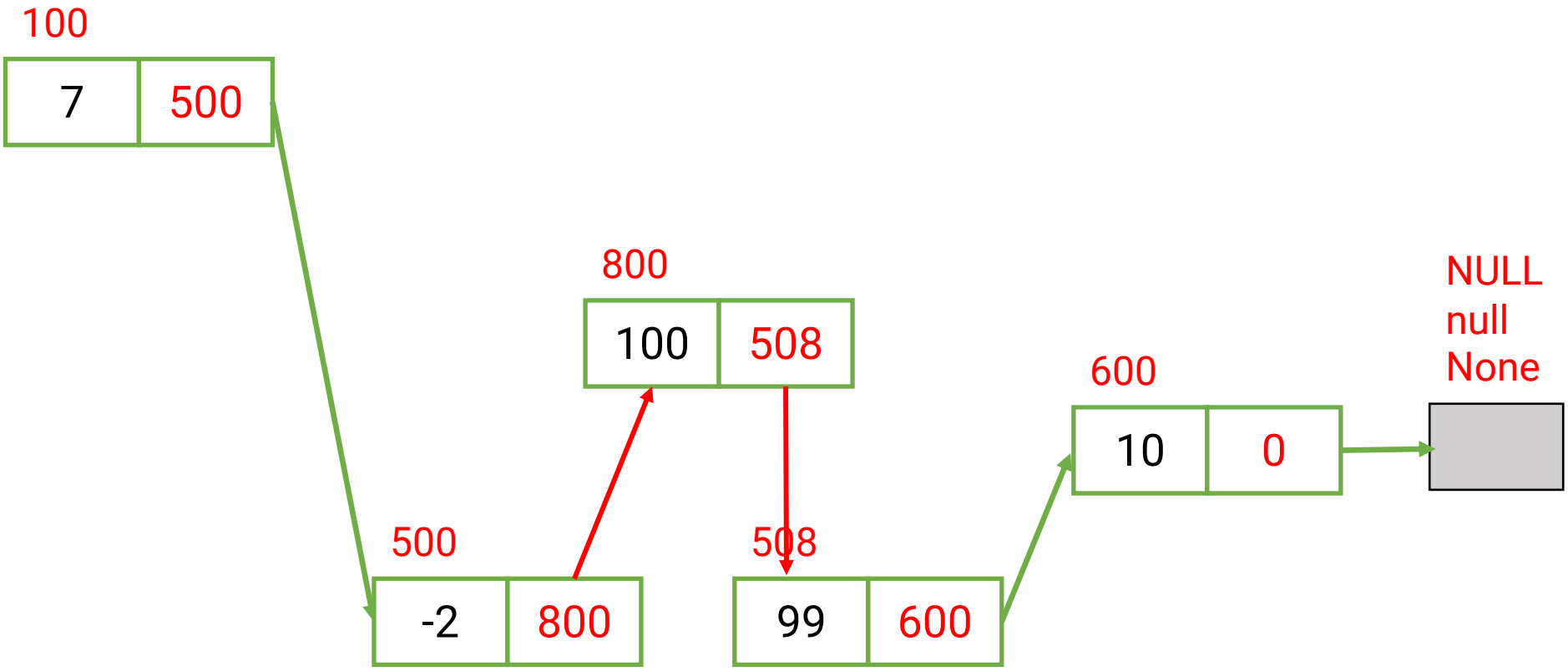
insertAfter(lst, -2, 100) – Thực hiện



insertAfter(lst, -2, 100) – Thực hiện



insertAfter(lst, -2, 100) – Thực hiện



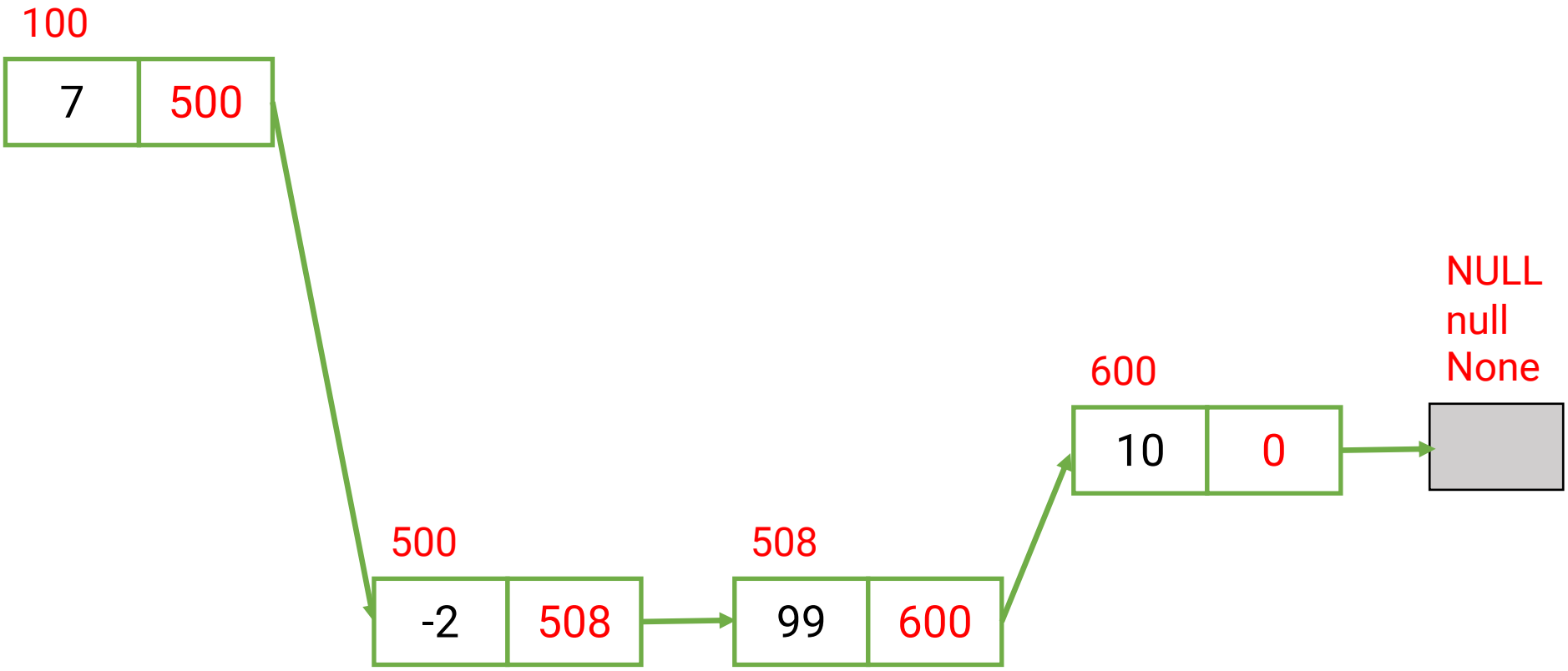
Time complexity: O(1)

Điểm mạnh của linked list

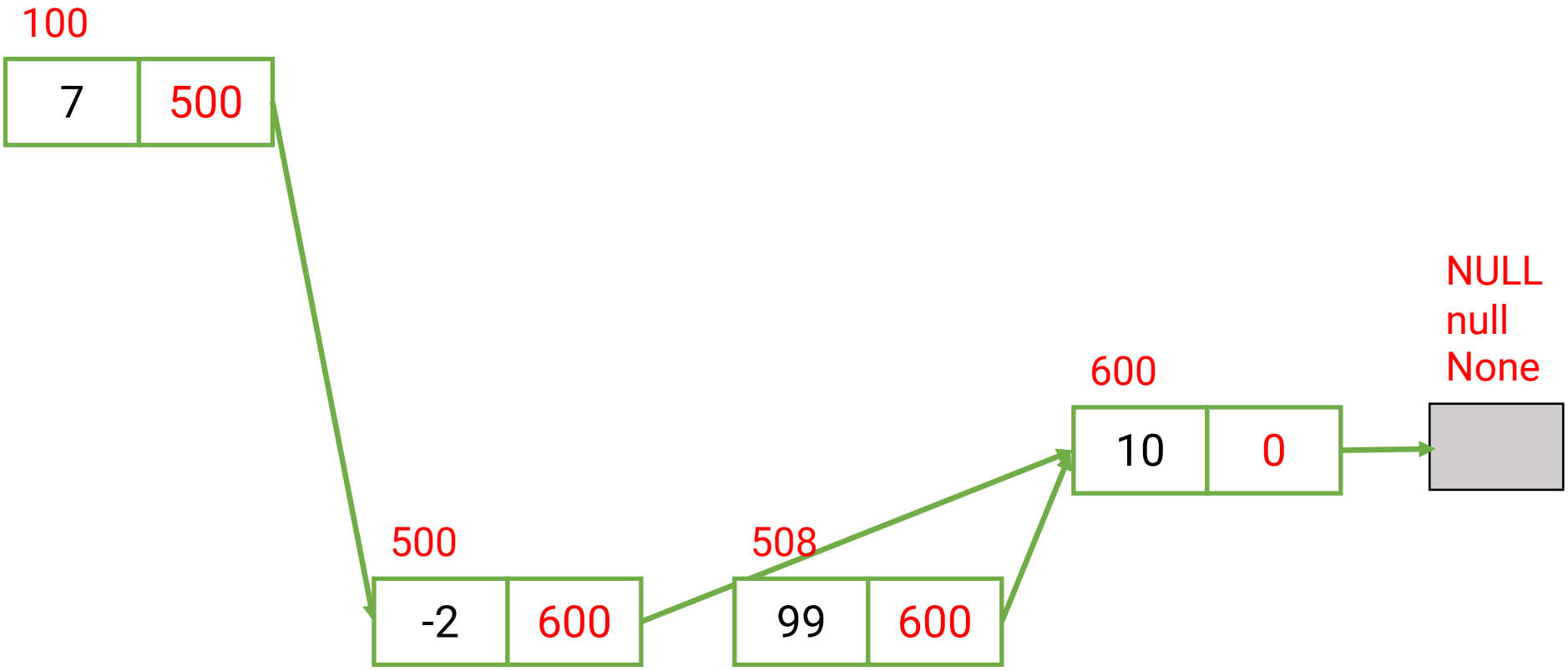
- Độ phức tạp thời gian của thao tác insertAfter(lst, x, y) : **O(1)**.

- Tương tự, độ phức tạp thời gian của thao tác remove(lst, x) : **O(1)**.
  - Xem các slide sau.

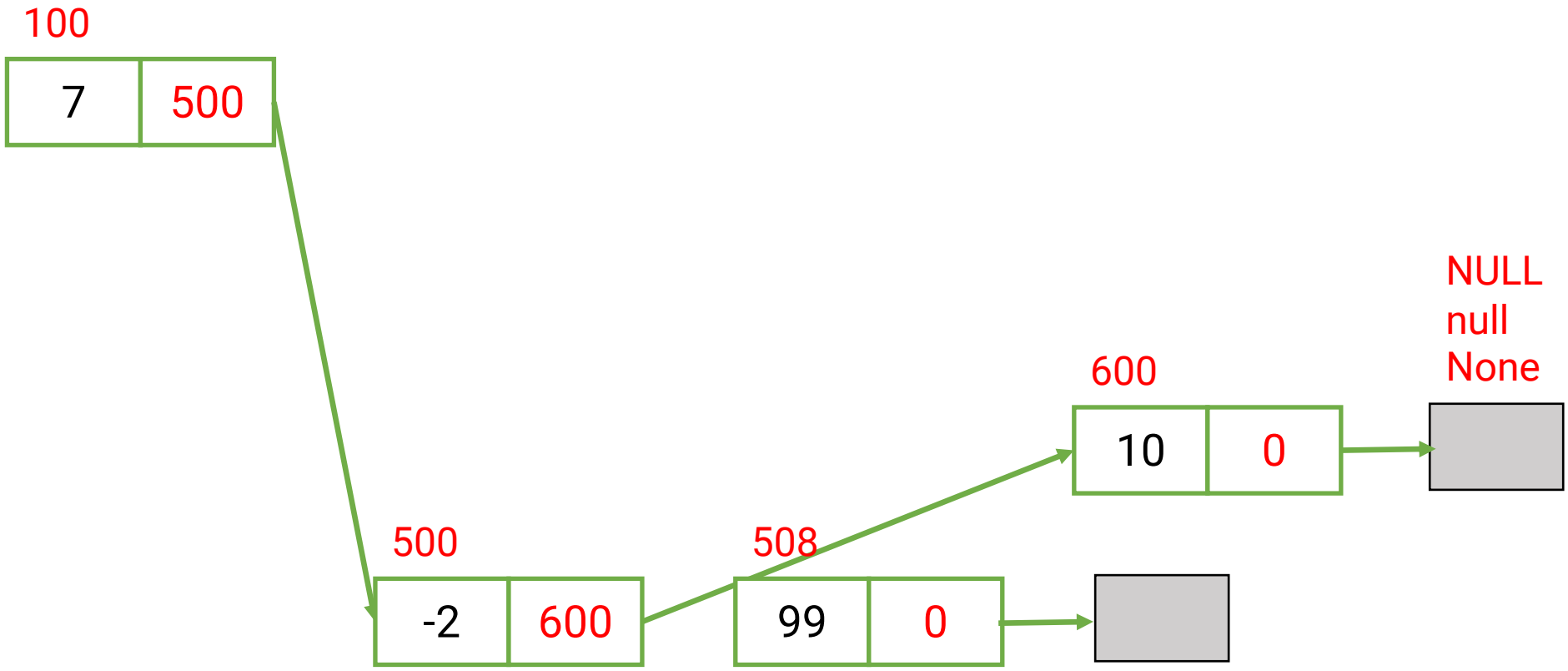
remove(lst, 99) – Thực hiện



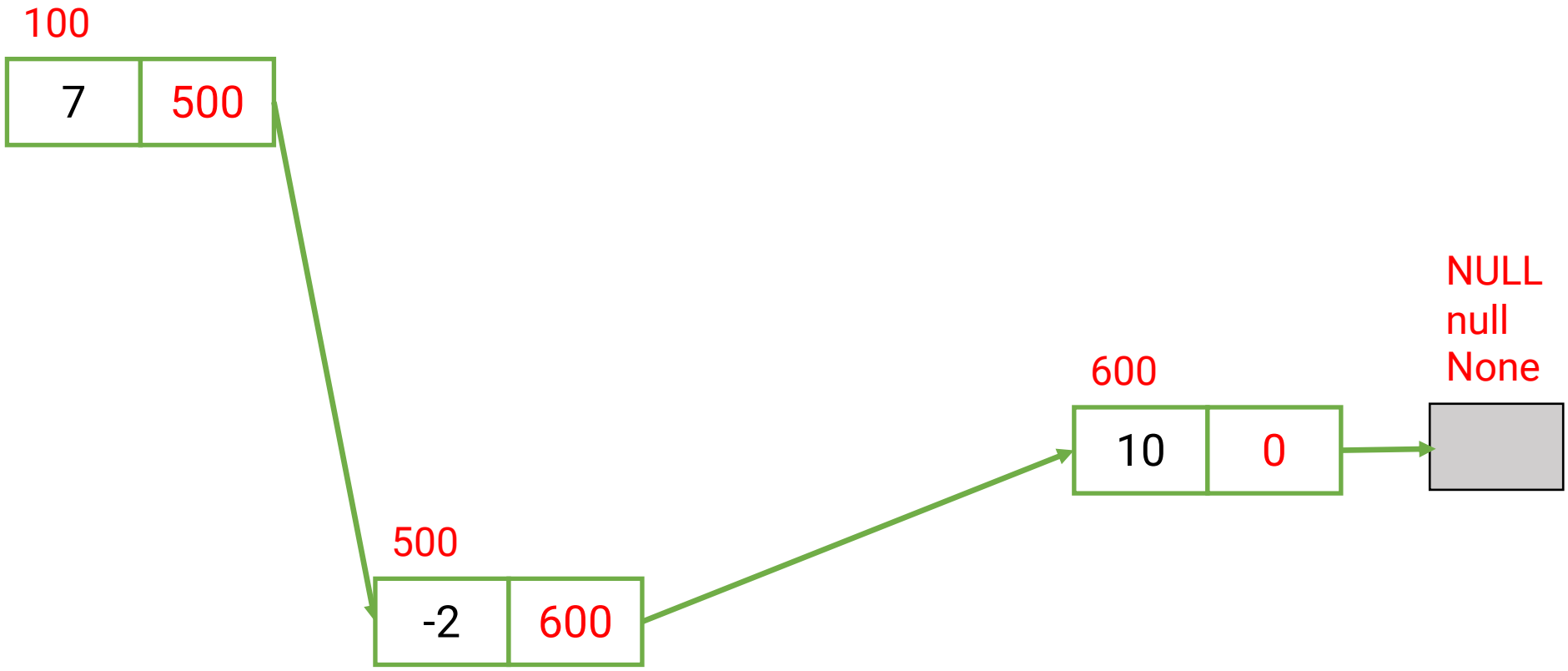
remove(lst, 99) – Thực hiện



remove(lst, 99) – Thực hiện



remove(lst, 99) – Thực hiện



Time complexity: O(1)

Quiz

- Có linked list lst, 1.000.000 phần tử
- Địa chỉ của phần tử thứ 0 trong linked list lst: 100
- Địa chỉ của phần tử thứ 6789 trong linked list lst: ?

Điểm yếu của linked list

- Độ phức tạp thời gian của thao tác lấy / cập nhật giá trị phần tử thứ i trong linked list: **O(n)**.

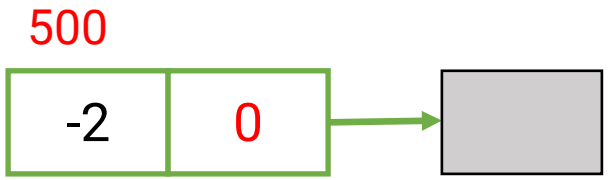
Linked List vs Array

	Array	Linked List
Cách lưu trữ	Liên tiếp	Không liên tiếp
Thêm phần tử x	O(n)	**O(1)**
Xóa phần tử x	O(n)	**O(1)**
Tìm phần tử thứ i	**O(1)**	O(n)
Tìm phần tử x	O(n)	O(n)
Chi phí bộ nhớ	data	**data, next**

Sử dụng linked list khi insert/remove phần tử nhiều lần

struct/class Node

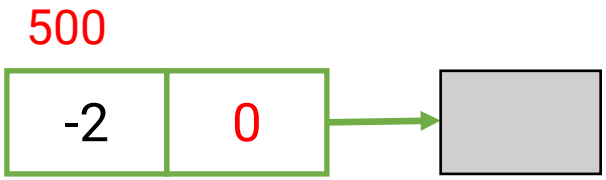
- Một node trong linked list chứa 2 thông tin sau:
  - **data**: dữ liệu lưu trong node đó: số nguyên, số thực, chuỗi, Fraction, Product…
  - **next**: node tiếp theo trong linked list.



struct Node (C++)

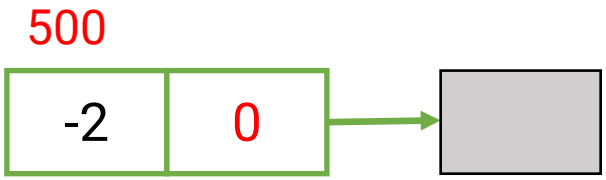
```
struct Node
{
  int data;
  Node *next;
};
```

```
Node* createNode(int x)
{
  Node *p = new Node;
  p->data = x;
  p->next = NULL;
  return p;
}
```



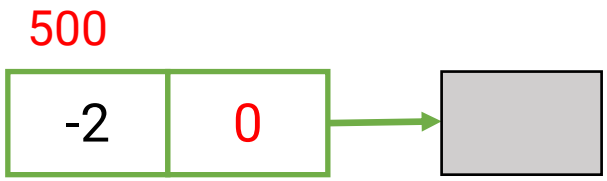
class Node (Java)

```java
class Node{
    public int data;
    public Node next;

    public Node(){
        data = 0;
        next = null;
    }
    public Node(int x){
        data = x;
        next = null;
    }
}
```



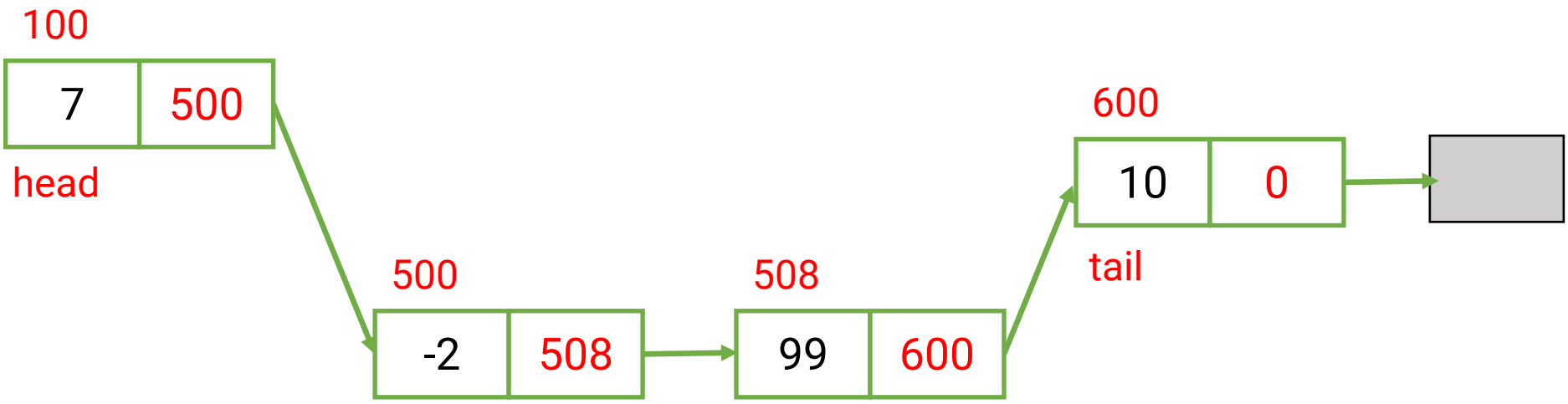
struct/class Node (Python)

```
class Node:
  def __init__(self, x = None):
    self.data = x
    self.next = None
```



struct/class LinkedList

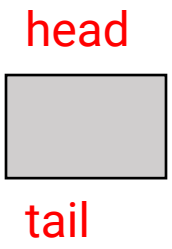
- Lưu thông tin của một linked list.
  - **head**: node đầu tiên trong linked list.
  - **tail**: node cuối cùng trong linked list.



struct LinkedList (C++)

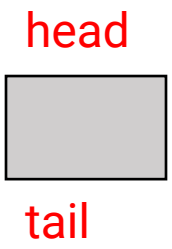
```
struct LinkedList
{
  Node *head;
  Node *tail;
};
```

```
void init(LinkedList &lst)
{
  lst.head = NULL;
  lst.tail = NULL;
}
```



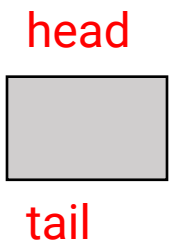
class LinkedList (Java)

```
class LinkedList{
    private Node head;
    private Node tail;
    public LinkedList(){
        head = null;
        tail = null;
    }
}
```



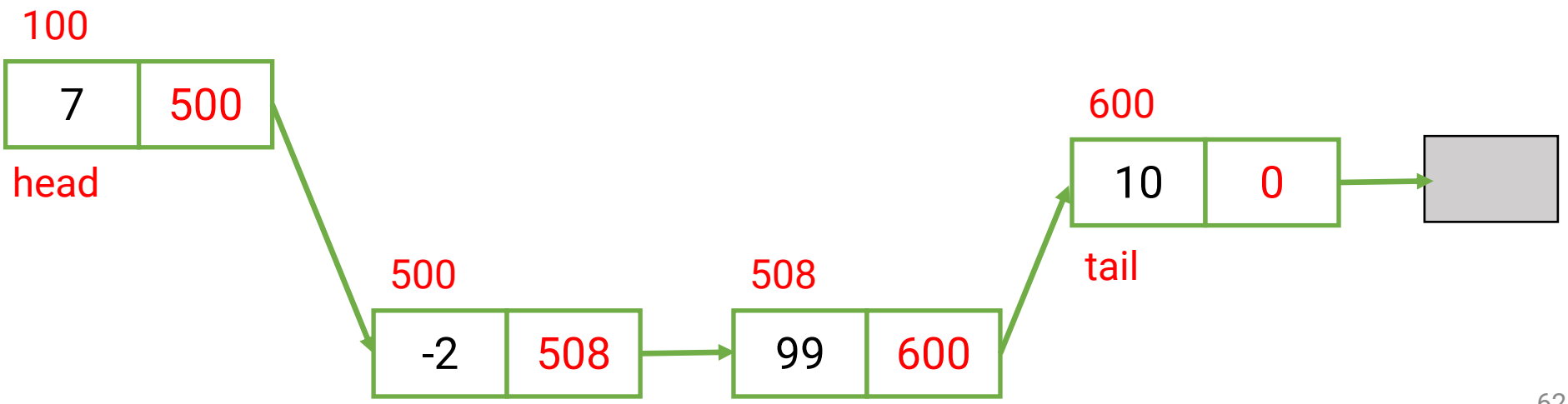
class LinkedList (Python)

```
class LinkedList:
  def __init__(self):
    self.head = None
    self.tail = None
```



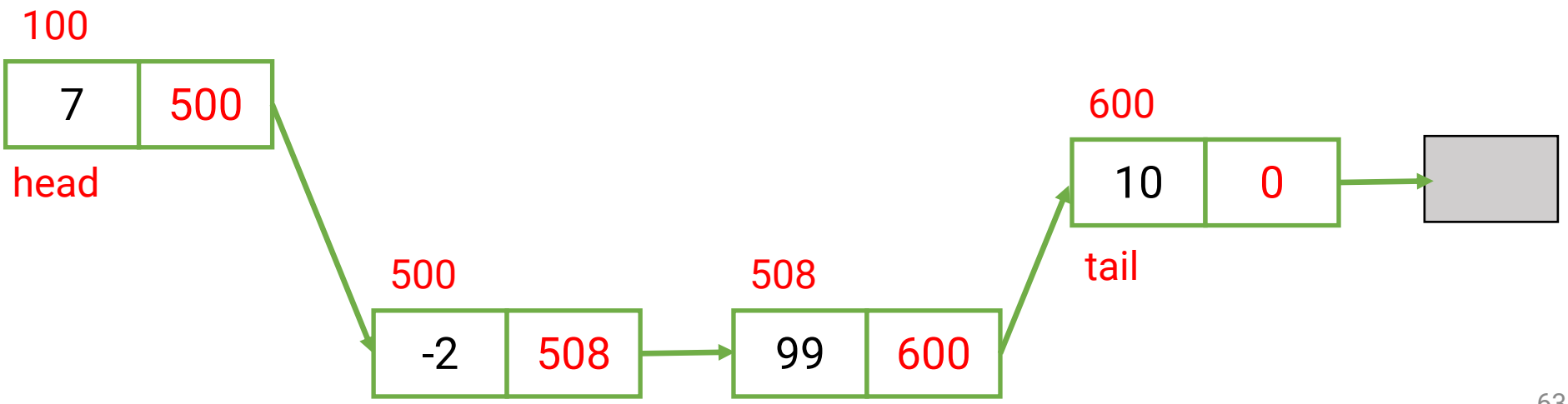
Quiz

```
print(lst.head.data)

print(lst.head.next.data)

print(lst.head.next.next.data)

print(lst.head.next.next.next.data)

print(lst.head.next.next.next.next.next.next.data)
```



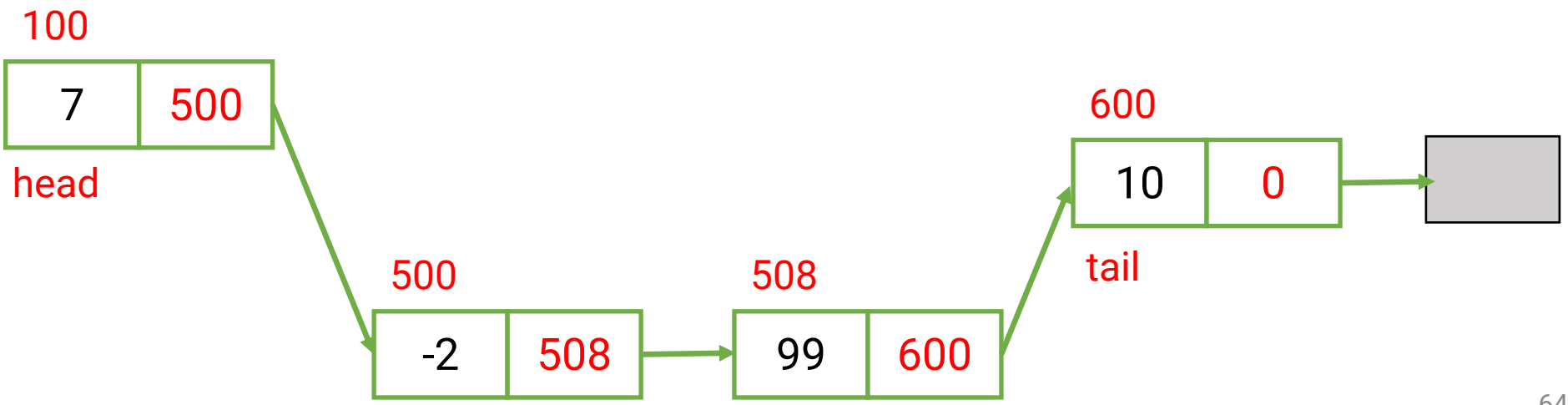
Quiz

```
p = lst.head
print(p.data)

p = p.next
print(p.data)
```



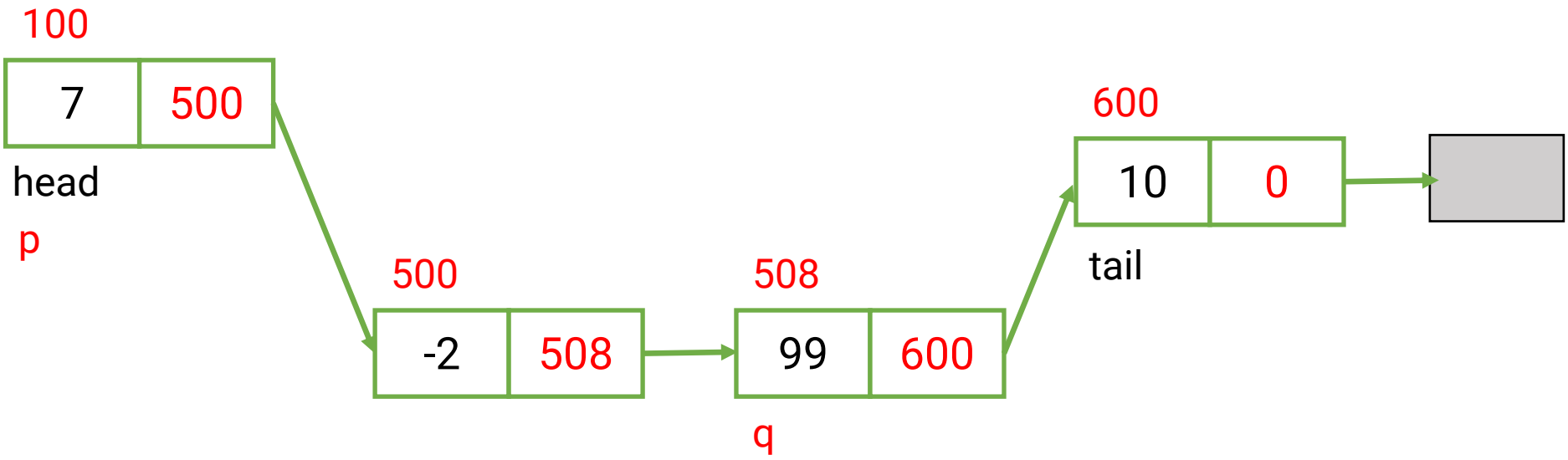
Quiz

```
q = p.next.next
print(q.data)

p.next = q

p.next = q.next
```



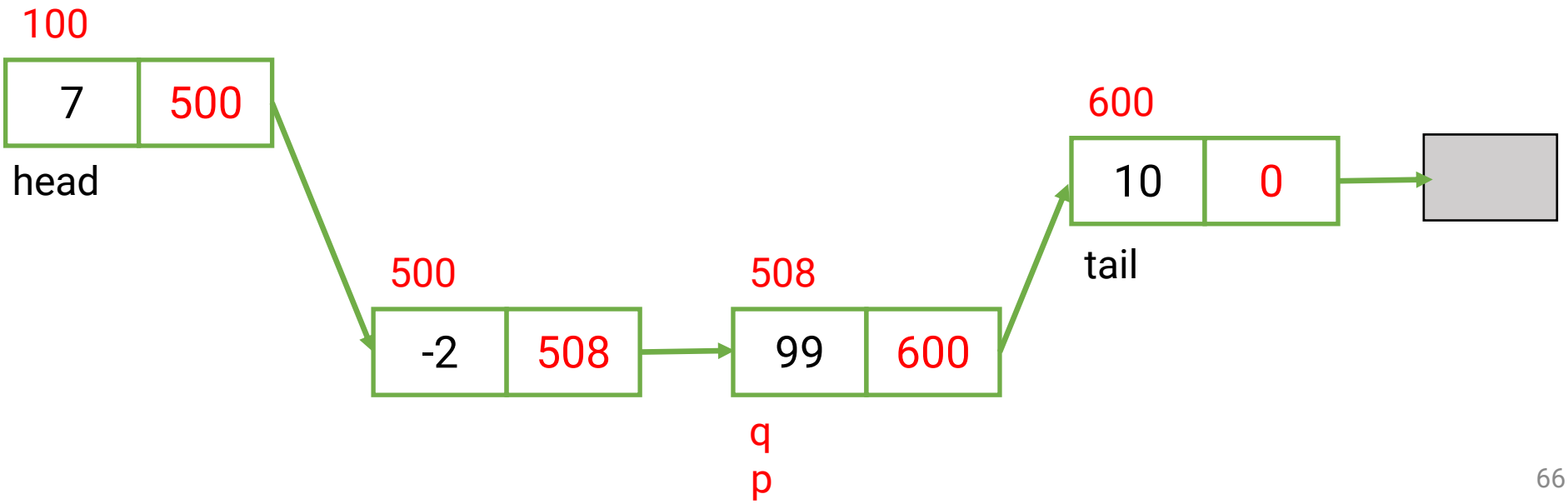
Khi muốn duyệt qua 1 node

```
p = lst.head
q = lst.head.next.next

p = q
```



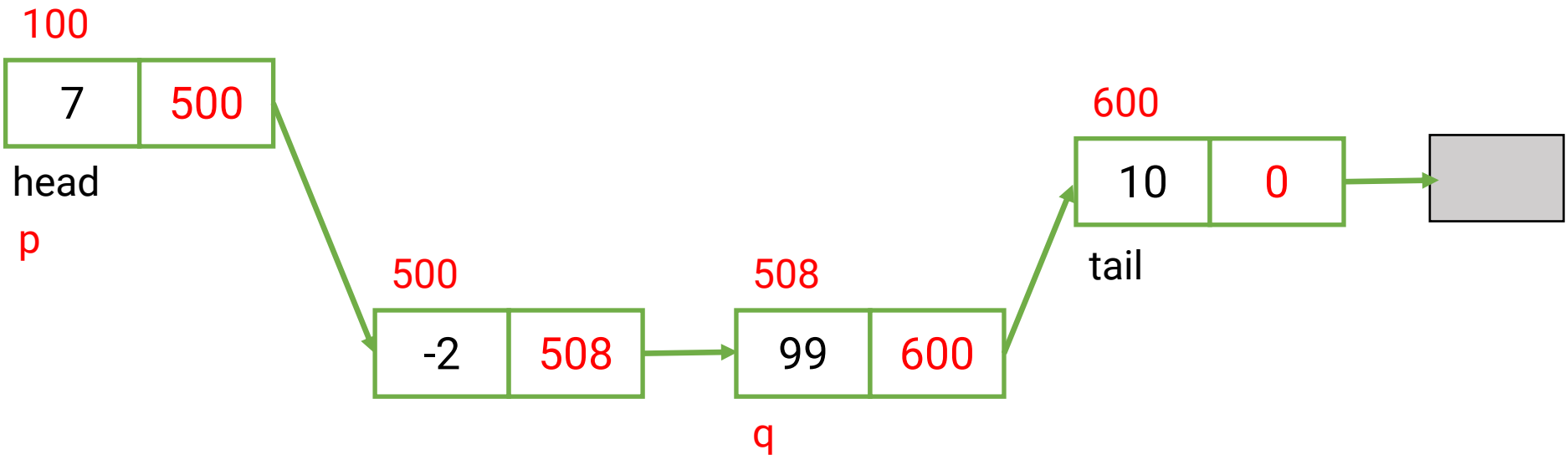
Khi muốn duyệt qua 1 node

```
p = lst.head
q = lst.head.next.next

p = q
```



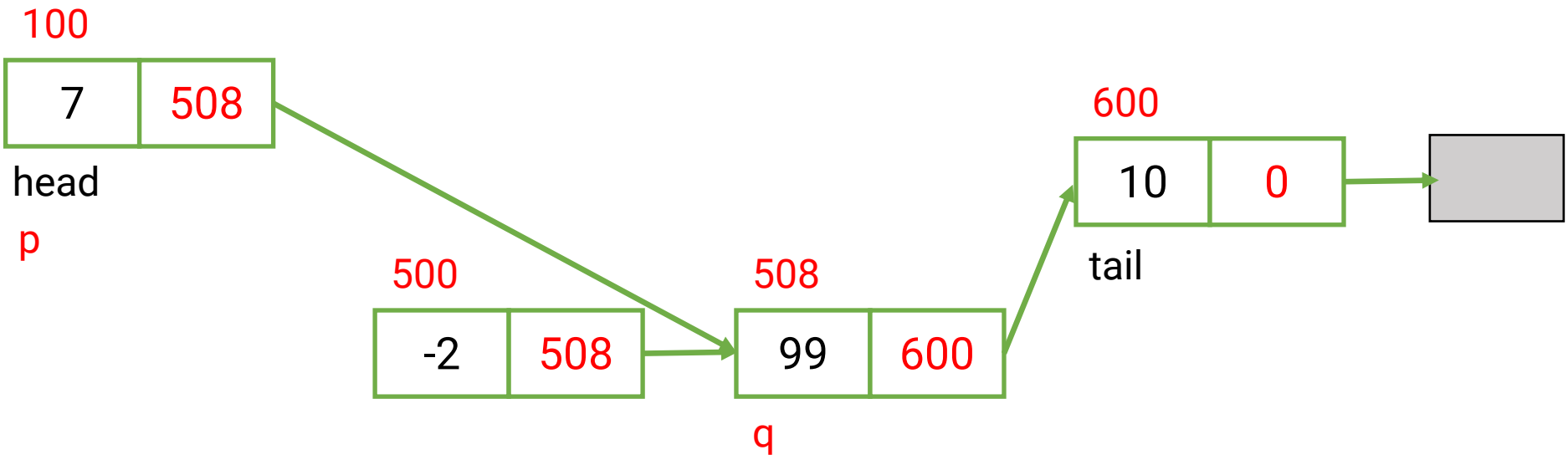
Khi muốn sửa liên kết trong list

```
p = lst.head
q = lst.head.next.next

p.next = q
```



Khi muốn sửa liên kết trong list

```
p = lst.head
q = lst.head.next.next

p.next = q
```



Các kĩ thuật xử lí trên linked list

- **init():** kĩ thuật khởi tạo list rỗng (không có phần tử nào).
- **traverse():** kĩ thuật duyệt qua các phần tử trong list.
  - In các phần tử trong list.
  - Tìm kiếm 1 phần tử.
  - Tìm phần tử nhỏ nhất, lớn nhất.
  - Đếm số lượng.
  - Tính tổng.

Các thao tác cơ bản trên linked list

- **insertHead(x):** kĩ thuật thêm phần tử x vào đầu list.
- **insertAfter(x, y):** kĩ thuật thêm phần tử y vào sau x.
- **insertTail(x):** kĩ thuật thêm phần tử x vào cuối list.

- **removeHead():** kĩ thuật xóa phần tử đầu ra khỏi list.
- **removeAfter(x, y):** kĩ thuật xóa 1 phần tử y nằm sau 1 phần tử x ra khỏi list.
- **removeTail():** kĩ thuật xóa phần tử cuối ra khỏi list.

BT1 – SỐ NHỎ NHẤT

- Cho danh sách liên kết đơn các số nguyên tìm số nhỏ nhất trong danh sách.

BT1 – Gợi ý – Xử lí chính

1. Khởi tạo linked list lst rỗng.
2. Sử dụng vòng lặp while(1)
   a. Đọc số nguyên x.
   b. Nếu x = 0, kết thúc vòng lặp.
   c. Ngược lại, gọi hàm **insertTail(x)** để thêm x vào cuối linked list.
3. Gọi hàm **min()** để tìm node p có data nhỏ nhất trong linked list.
4. In data của p.
5. Gọi hàm removeAll() để xóa linked list. (C++)

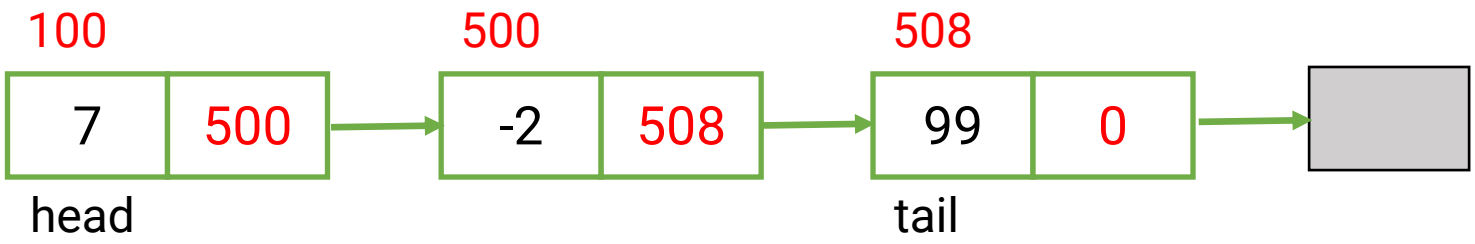
BT1 – Gợi ý – Chuẩn bị

1. Khai báo struct/class Node.
2. Khai báo struct/class LinkedList.
3. Cài đặt hàm insertTail().
4. Cài đặt hàm min().
5. Cài đặt hàm removeAll(). (C++)

BT1 – Gợi ý – Hàm insertTail(x)

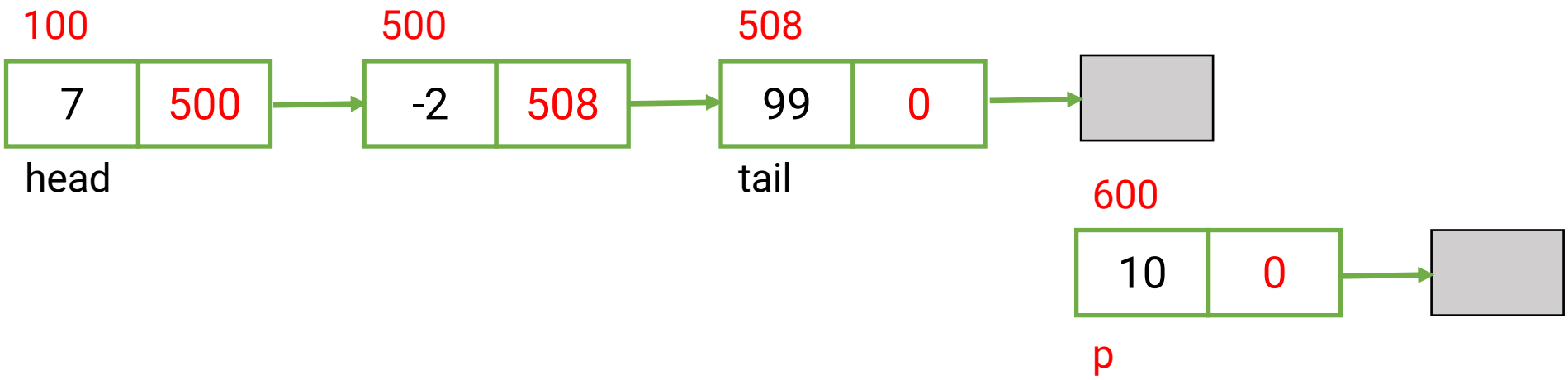
1. Tạo node cho số nguyên x.
2. Nếu list rỗng (head là NULL / null / None), cập nhật head, tail là p.
3. Ngược lại, list khác rỗng.
   a. Next của tail là p.
   b. Cập nhật lại, tail là p.

Minh họa insertTail(10)



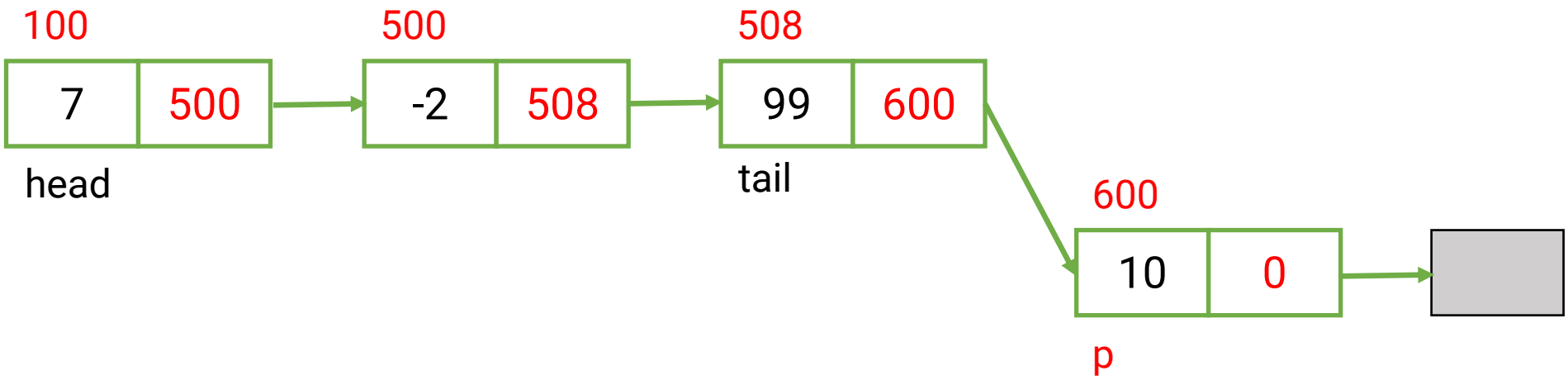
- Giả sử ta đang có linked list gồm 3 số: 7, -2, 99.
- Ta cần thêm x = 10 vào cuối list (vào sau số 99).

Minh họa insertTail(10)



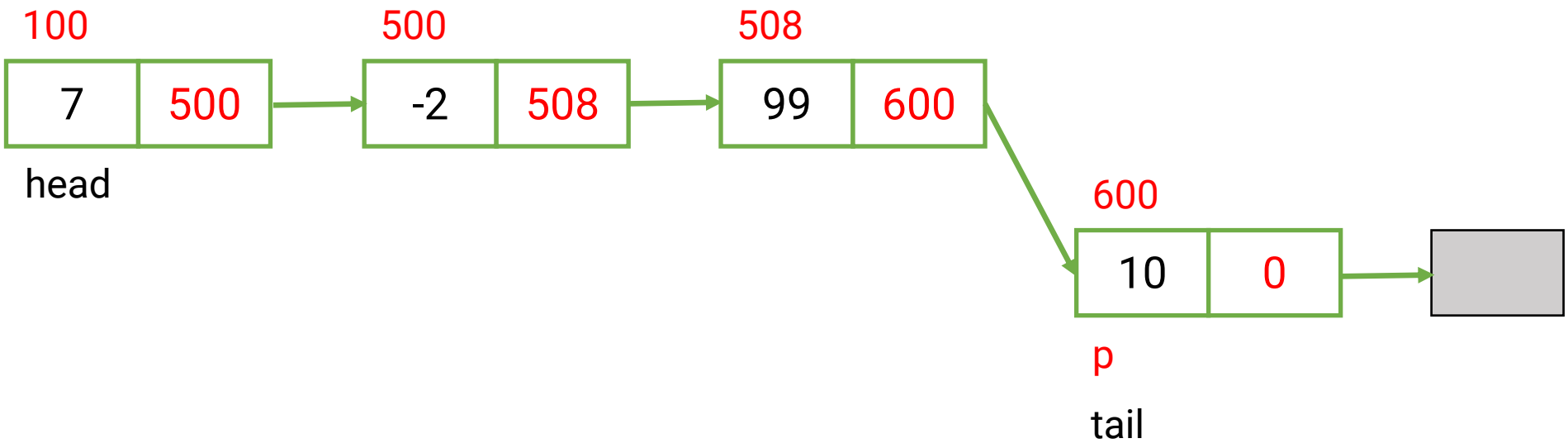
- Tạo node p cho số 10:
  - p.data = 10
  - p.next = None

Minh họa insertTail(10)



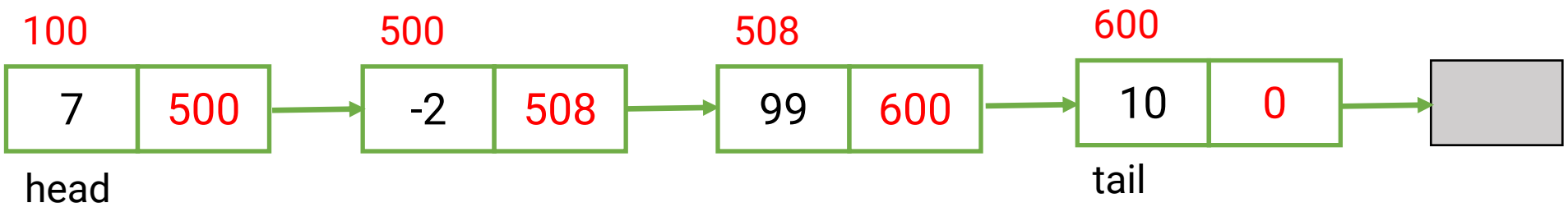
- Cần thêm node 10 vào sau node 99.
- Mà tail đang ở node 99.
- Do đó, chỉ cần tail.next = p là thêm thành công.

Minh họa insertTail(10)



- Cần thêm node 10 vào sau node 99.
- Mà tail đang ở node 99.
- Do đó, chỉ cần tail = p là thêm thành công.

Minh họa insertTail(10)



insertTail(lst, x) (C++)

```
void insertTail(LinkedList &lst, int x)
{
    Node *p = createNode(x);
    if(lst.head == NULL)
    {
        lst.head = lst.tail = p;
    }
    else
    {
        lst.tail->next = p;
        lst.tail = p;
    }
}
```

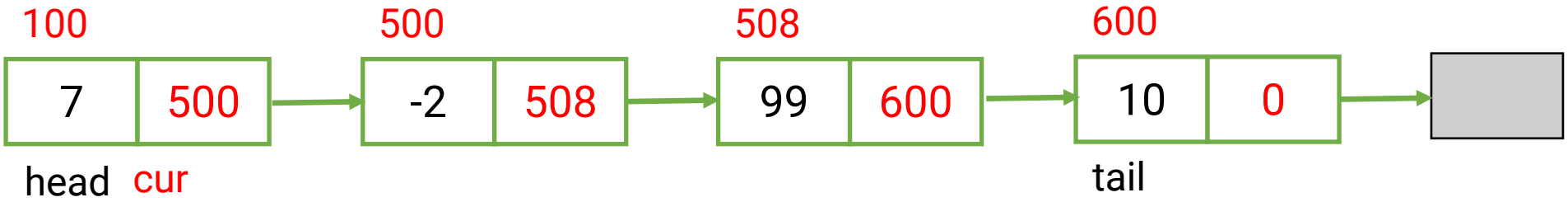
lst.insertTail(x) (Java)

```
class LinkedList{
    private Node head;
    private Node tail;

    public void insertTail(int x){
        Node p = new Node(x);
        if(head == null){
            head = tail = p;
        }
        else{
            tail.next = p;
            tail = p;
        }
    }
}
```

lst.insertTail(x) (Python)

```
class LinkedList:
    def insertTail(self, x):
        p = Node(x)
        if(self.head == None):
            self.head = self.tail = p
        else:
            self.tail.next = p
            self.tail = p
```

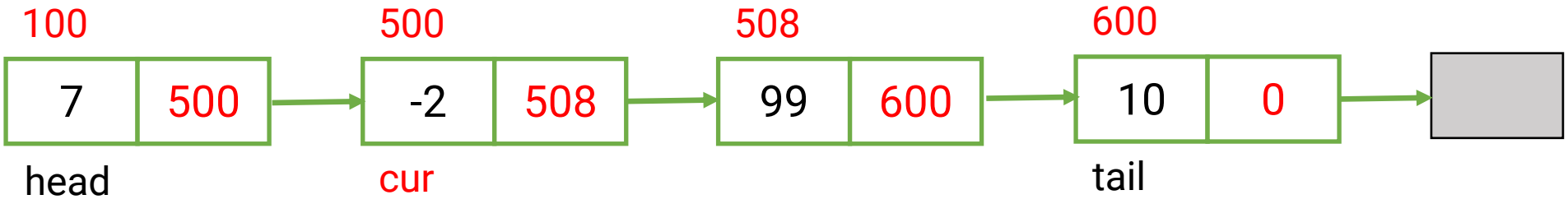
Minh họa duyệt qua linked list



- cur bắt đầu từ head: cur = head

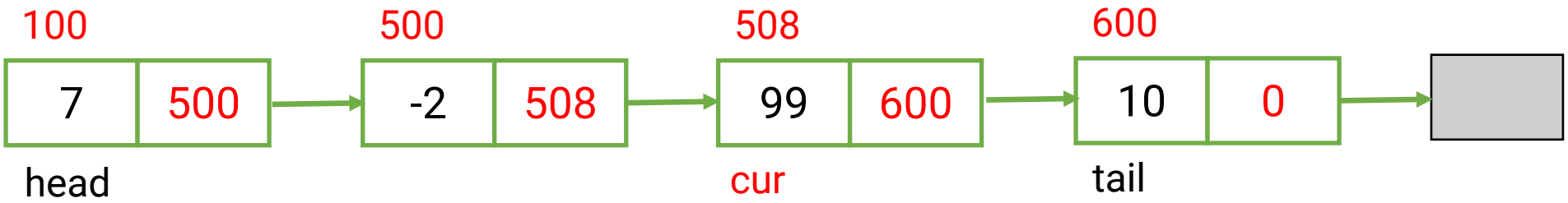
Kĩ thuật traverse()

Minh họa duyệt qua linked list



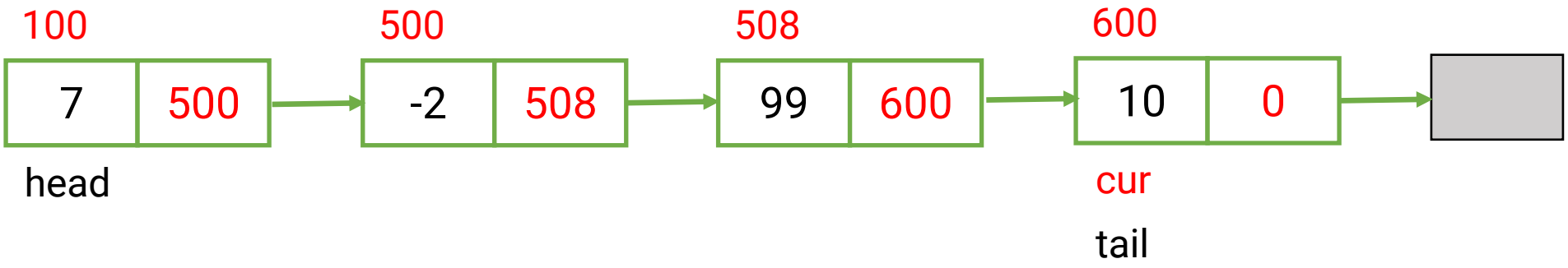
- cur di chuyển đến node kế tiếp: cur = cur.next

Minh họa duyệt qua linked list



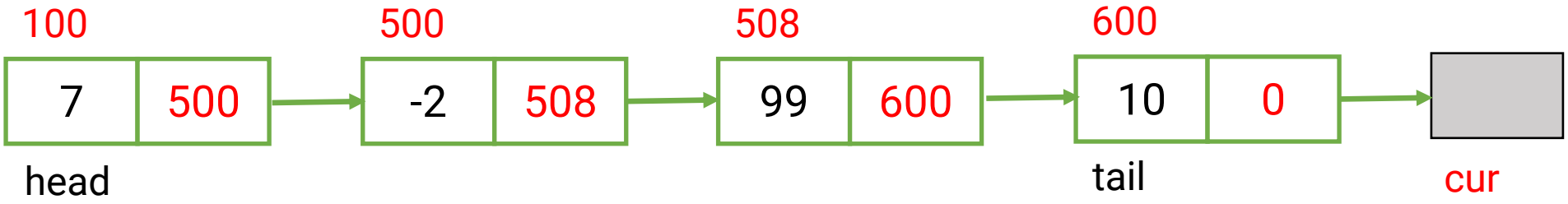
- cur di chuyển đến node kế tiếp: cur = cur.next

Minh họa duyệt qua linked list



- cur di chuyển đến node kế tiếp: cur = cur.next

Minh họa duyệt qua linked list



- cur di chuyển đến node kế tiếp: cur = cur.next
- Khi nào cur di chuyển đến node None thì dừng lại.

BT1 – Gợi ý – Hàm min()

- **Dùng kĩ thuật đặt lính canh.**

1. Nếu list rỗng (head là NULL / null / None), trả về NULL / null / None.
2. Khởi tạo giá trị nhỏ nhất là node đầu tiên: ans = head.
3. Khởi tạo cur = head.
4. Vòng lặp cur khác NULL / null / None.
    a. Nếu cur.data < ans.data, cập nhật lại ans = cur.
    b. Di chuyển cur đến node kế tiếp.
5. Trả về node ans.

min(lst) (C++)

```
Node* min(LinkedList lst)
{
    if(lst.head == NULL)
    {
        return NULL;
    }
    Node *ans = lst.head;
    Node *cur = lst.head;
    while(cur != NULL)
    {
        if(cur->data < ans->data)
        {
            ans = cur;
        }
        cur = cur->next;
    }
    return ans;
}
```

lst.min() (Java)

```java
class LinkedList{
    public Node min(){
        if(head == null){
            return null;
        }
        Node ans = head;
        Node cur = head;
        while(cur != null){
            if(cur.data < ans.data){
                ans = cur;
            }
            cur = cur.next;
        }
        return ans;
    }
}
```

lst.min() (Python)

```
class LinkedList:
    def min(self):
        if(self.head == None):
            return None
        ans = self.head
        cur = self.head
        while(cur != None):
            if(cur.data < ans.data):
                ans = cur
            cur = cur.next
        return ans
```

Gợi ý hàm removeAll() (C++)

- Node* p trong C++ được gọi là 1 con trỏ (pointer).
- Trong C++, khi sử dụng con trỏ, developer phải lo việc cấp phát (new) và hủy bỏ (delete) con trỏ để tránh lãng phí vùng nhớ.
- Trong Java và Python, garbage collector (bộ thu gom rác) lo việc cấp phát và hủy bỏ này.
- Do đó, C++ cần cài đặt thêm hàm removeAll() còn Java và Python thì không.

Gợi ý hàm removeAll() (C++)

```
void removeAll(LinkedList &lst)
{
  while(lst.head != NULL)
  {
    Node * cur = lst.head;
    lst.head = lst.head->next;
    delete cur;
  }
  lst.tail = NULL;
}
```

```
// Java: Do nothing here!!!
```

```
# Python: Do nothing here!!!
```

BT2 – LIỆT KÊ SV DƯỚI TRUNG BÌNH

- Cho danh sách liên kết các điểm dưới dạng số thực. Hãy in ra danh sách các điểm dưới trung bình (điểm dưới 5.0).

BT2 – Gợi ý – Xử lí chính

1. Khởi tạo linked list lst rỗng.
2. Dùng vòng lặp while(1):
   a. Đọc vào điểm trung bình x.
   b. Nếu x = -1, kết thúc vòng lặp.
   c. Ngược lại, gọi hàm **insertTail(x)** để thêm x vào cuối linked list.
3. Gọi hàm **printBelowAvg()** liệt kê danh sách sinh viên điểm dưới 5.
4. Gọi hàm removeAll() để xóa linked list. (C++)

BT2 – Gợi ý – Chuẩn bị

1. Khai báo struct/class Node.
2. Khai báo struct/class LinkedList.
3. Cài đặt hàm insertTail().
4. Cài đặt hàm printBelowAvg().
5. Cài đặt hàm removeAll(). (C++)

BT3 – ĐẾM SỐ NGUYÊN TỐ

- Cho lần lượt các số nguyên dương, hãy tạo danh sách liên kết đơn từ các số đó và đếm xem có bao nhiêu số nguyên tố trong danh sách liên kết đơn mới tạo.

BT3 – Gợi ý – Xử lí chính

1. Khởi tạo linked list lst rỗng.
2. Sử dụng vòng lặp while(1)
   a. Đọc số nguyên x.
   b. Nếu x = 0, kết thúc vòng lặp.
   c. Ngược lại, gọi hàm **insertTail(x)** để thêm x vào cuối linked list.
3. Gọi hàm **countPrimes()** để đếm các số nguyên tố trong linked list.
4. In kết quả.
5. Gọi hàm removeAll() để xóa linked list. (C++)

BT3 – Gợi ý – Chuẩn bị

1. Khai báo struct/class Node.
2. Khai báo struct/class LinkedList.
3. Cài đặt hàm insertTail().
4. Cài đặt hàm countPrimes().
5. Cài đặt hàm removeAll(). (C++)

BT4 – CHÈN SỐ THỨ TỰ

- Cho danh sách liên kết đơn các số nguyên. Chèn trước mỗi phần tử số thứ tự của nó trong danh sách ban đầu. Cho rằng số thứ tự bắt đầu bằng 1.
- In ra danh sách sau khi đã chèn.

BT4 – Gợi ý – Xử lí chính

1. Khởi tạo linked list lst rỗng.
2. Dùng vòng lặp while(1).
    1. Đọc số nguyên x.
    2. Nếu x = 0, kết thúc vòng lặp.
    3. Ngược lại, gọi hàm **insertTail(x)** để thêm x vào cuối linked list.
3. Gọi hàm **insertOrdNum()** để thêm từng số 1, 2, 3, 4,… vào trước mỗi số đã có trong list. Nếu p là NULL / null / None, list rỗng, in ra số 0.
4. Duyệt qua list, in từng số trong list.

Gợi ý hàm insertOrdNum()

- Cho dãy số 3 9 -5.
- Ta cần insert các số 1, 2, 3 vào trước.
- Như vậy, kết quả là **1** 3 **2** 9 **3** -5.

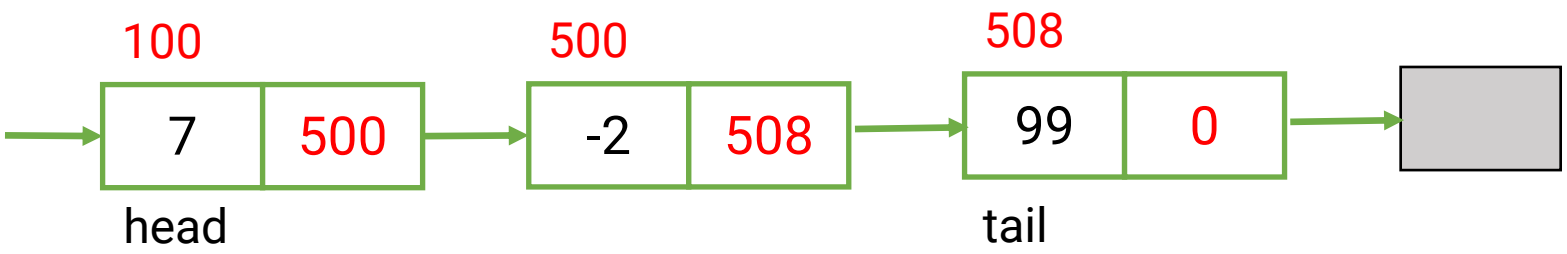
- Ở đây có 2 trường hợp cần xét:
  1. TH1: Thêm số 1 vào trước số 3 (vào trước head). TH1 này cần cập nhật lại head sau khi thêm 1 vào.
  2. TH2: Thêm số 2, 3 vào trước các số 9, -5. TH2 này không cần cập nhật lại head.

TH1: insert head

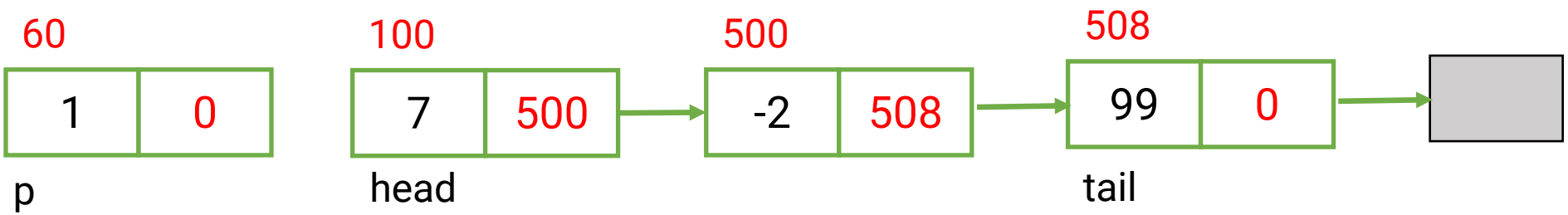
1. Tạo node p cho số x.
2. p.next = head.
3. head = p.

Minh họa insert head (x = 1)

1

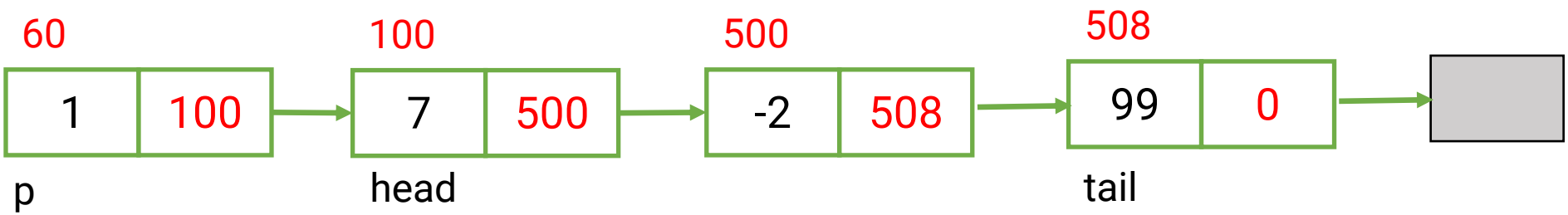


Minh họa insert head (x = 1)



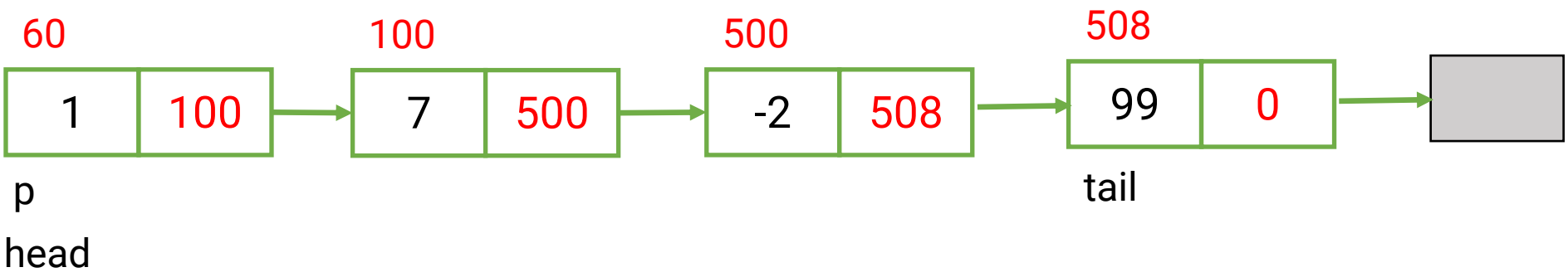
- Tạo node p cho x = 1.

Minh họa insert head (x = 1)



- p.next = head

Minh họa insert head (x = 1)



- Cập nhật lại: head = p.

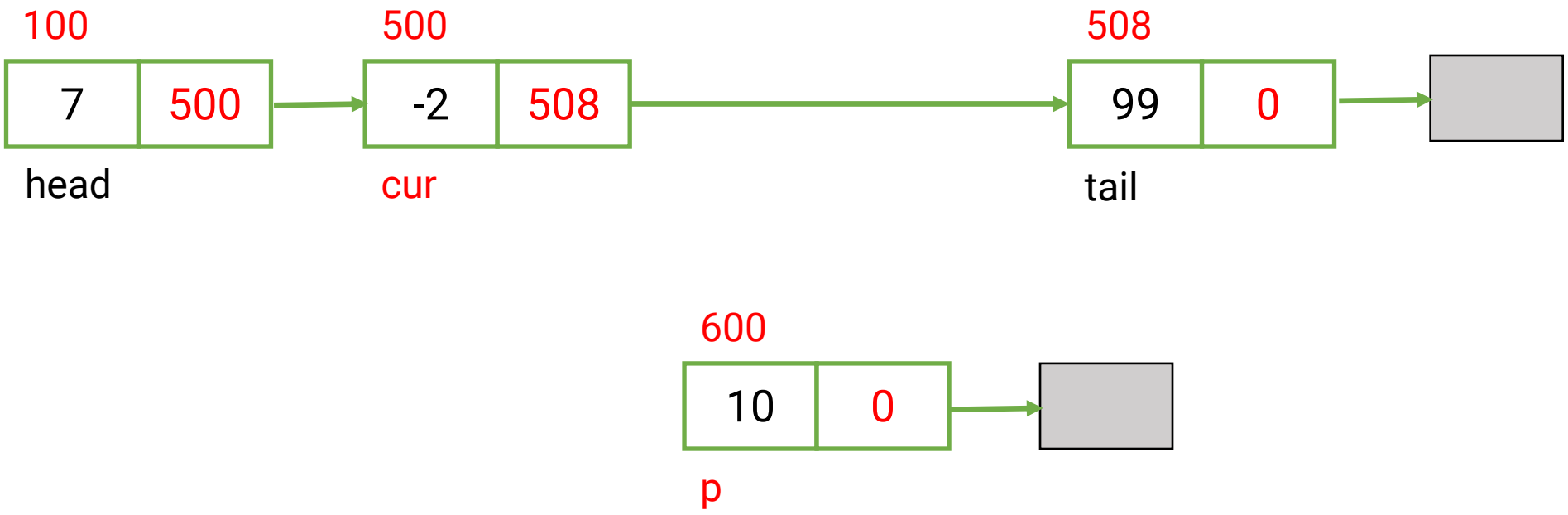
TH2: insert y after x

- Xét lại dãy số 3 9 -5.
- Ta đã thêm 1 vào trước 3: **1** 3 9 -5.
- Ta cần thêm 2, 3 trước 9, -5 để được: **1** 3 **2** 9 **3** -5.
  - Như vậy, ta có nhu cầu thêm **2 trước 9**, thêm **3 trước -5**.
- Tuy nhiên, node list chỉ có 1 chiều next, nên ta hãy suy nghĩ theo hướng ngược lại sẽ dễ dàng hơn: thêm **2 vào sau 3**, thêm **3 vào sau 9**.

Minh họa insert y after x (x = -2, y = 10)

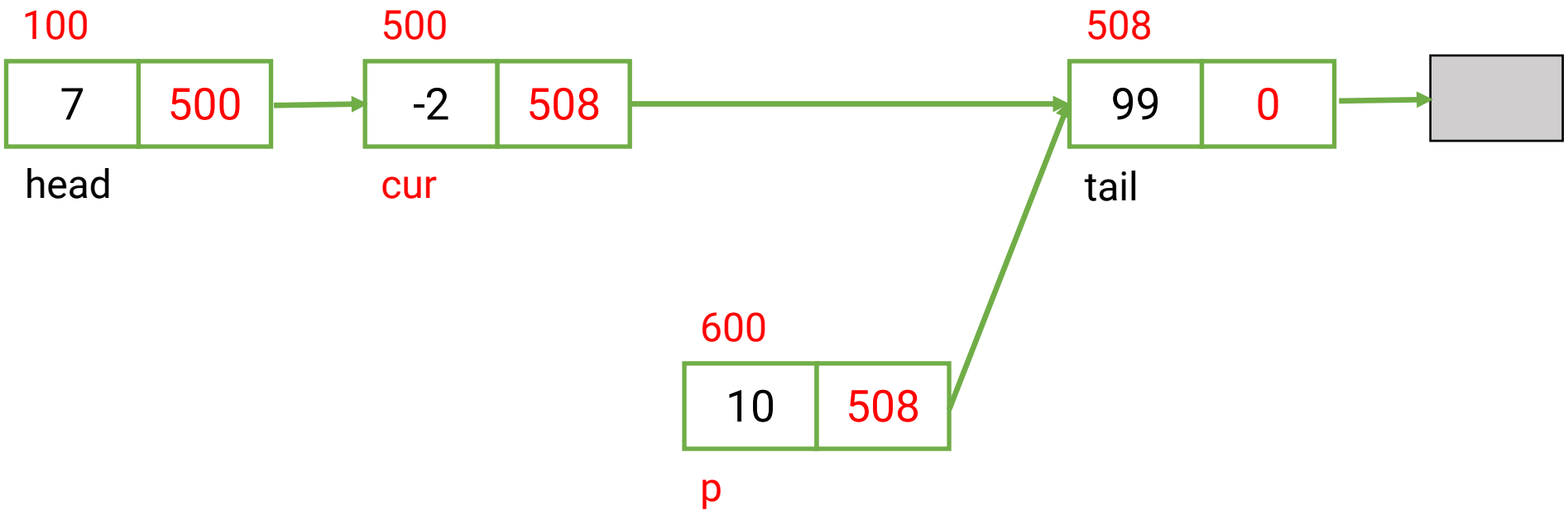
- Giả sử cur đang ở node -2 và ta dự định thêm 10 vào sau -2.

Minh họa insert y after x (x = -2, y = 10)



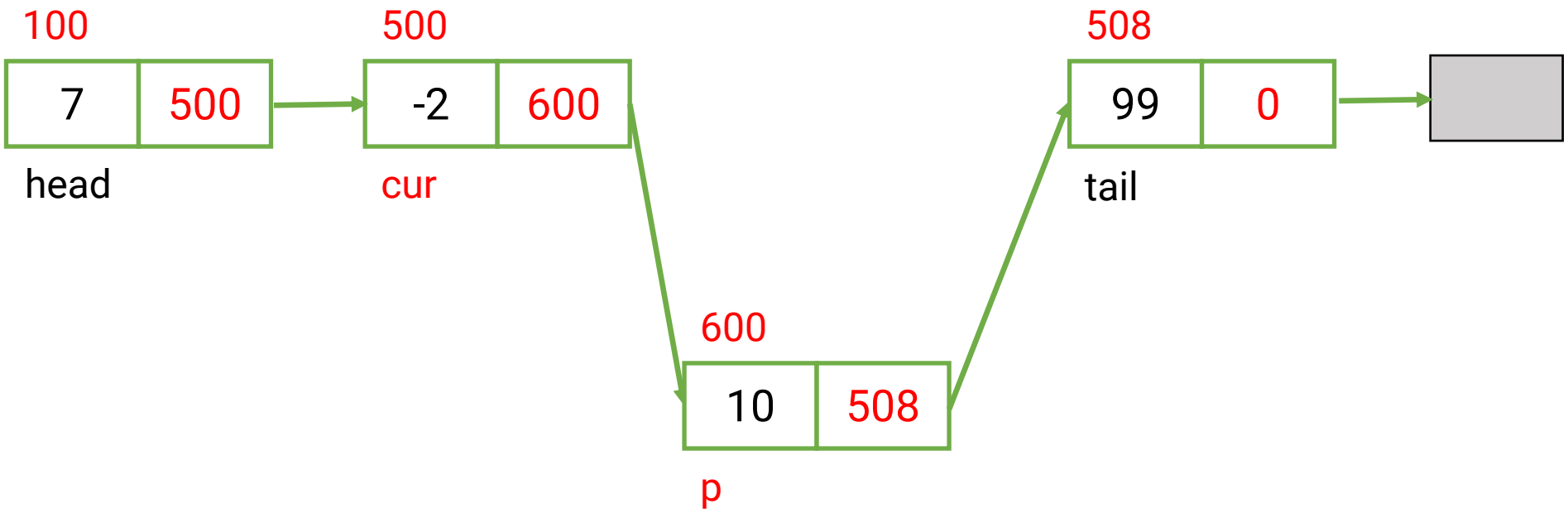
- Tạo node p cho số 10.

Minh họa insert y after x (x = -2, y = 10)



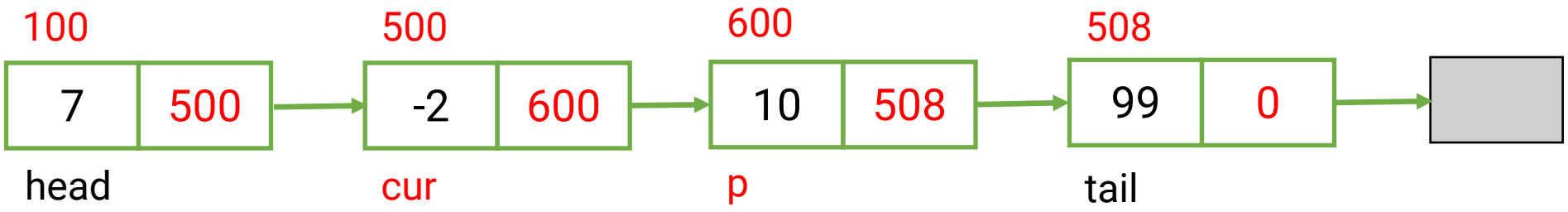
- Cho p.next = cur.next.

Minh họa insert y after x (x = -2, y = 10)



- Cập nhật lại, cur.next = p.

Minh họa insert y after x (x = -2, y = 10)



- DONE

BT4 – Gợi ý – Hàm insertOrdNum()

1. Tạo node p cho i = 1.
2. p.next = head.
3. head = p.

Gợi ý hàm insertOrdNum()

4. cur = p.next.
5. Vòng lặp cur.next is not NULL / null / None,
    a. Tạo node p cho i = 2.
    b. p.next = cur.next
    c. cur.next = p.
    d. cur = p.next
    e. Tăng i.

BT5 – XÓA TẬN CÙNG 5

- Cho danh sách liên kết đơn, hãy tìm và xóa những phần tử nào có tận cùng là 5.

Gợi ý xóa các số tận cùng là 5

- Do list chỉ có 1 chiều next, nên ta cần dùng 2 biến prev, cur đi sánh đôi với nhau.
- VD: dãy số 3 5 7 9.
  - cur = 3 thì prev = None.
  - cur = 5 thì prev = 3.
  - cur = 7 thì prev = 5.
  - cur = 9 thì prev = 7.
  - cur = NULL / null / None thì prev = 9.

Gợi ý xóa các số tận cùng là 5

- Để tạo được prev và cur sánh đôi ta làm như sau.

```
Node* prev = NULL;
Node* cur = lst.head;

while(cur != NULL)
{
  prev = cur;
  cur = cur->next;
}
```

```
prev = None
cur = self.head

while cur:

  prev = cur
  cur = cur.next
```

```
Node prev = null;
Node cur = head;
while(cur != null){
    prev = cur;
    cur = cur.next;
}
```

Gợi ý xóa các số tận cùng là 5

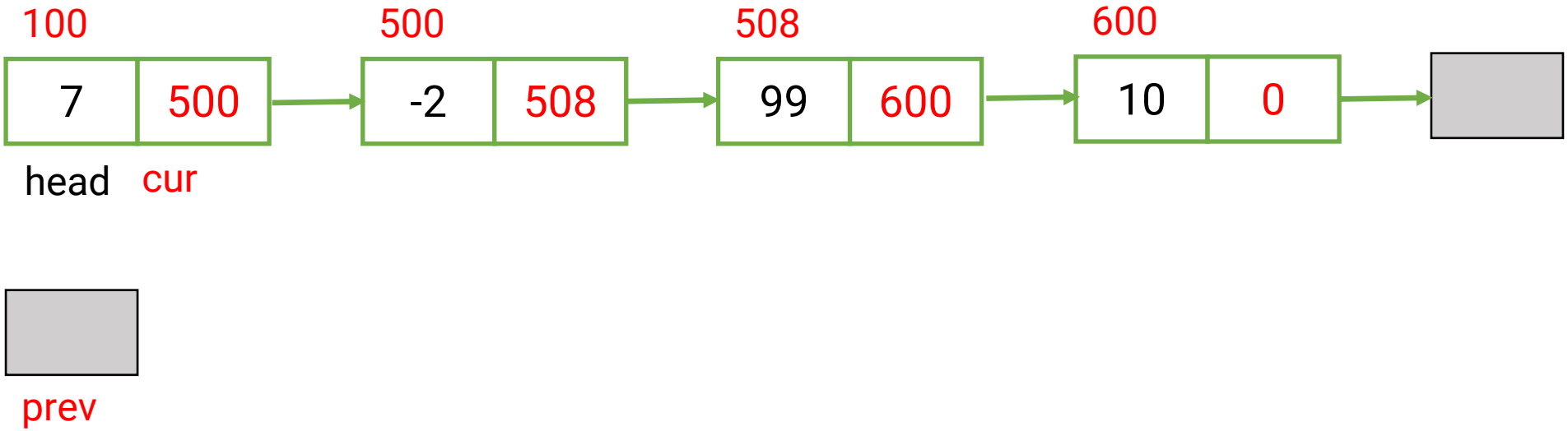
```
prev->next = cur->next;
delete cur;
cur = prev->next;
```

```
prev.next = cur.next

cur = prev.next
```

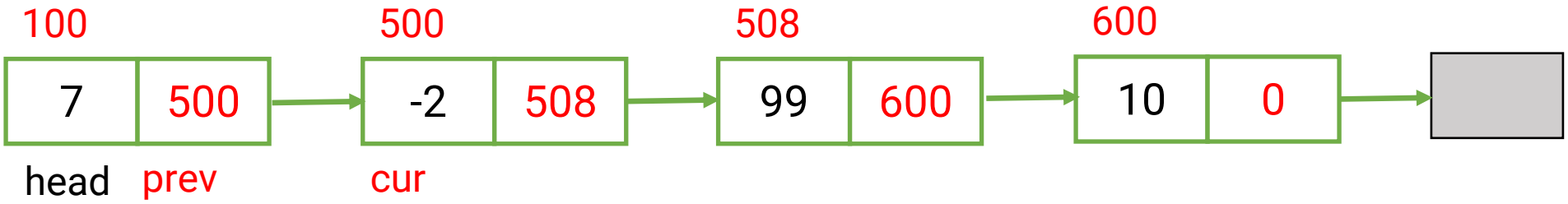
```
prev.next = cur.next;
cur = prev.next;
```

- Khi đã setup prev, cur rồi, thì việc xóa rất dễ dàng.
- Tuy nhiên:
  - Nhớ cập nhật lại head: head = cur.next, khi head.data tận cùng là 5 (nhận biết: prev == None).
  - Nhớ cập nhật lại tail: tail = prev, khi tail.data tận cùng là 5 (nhận biết: cur == tail).

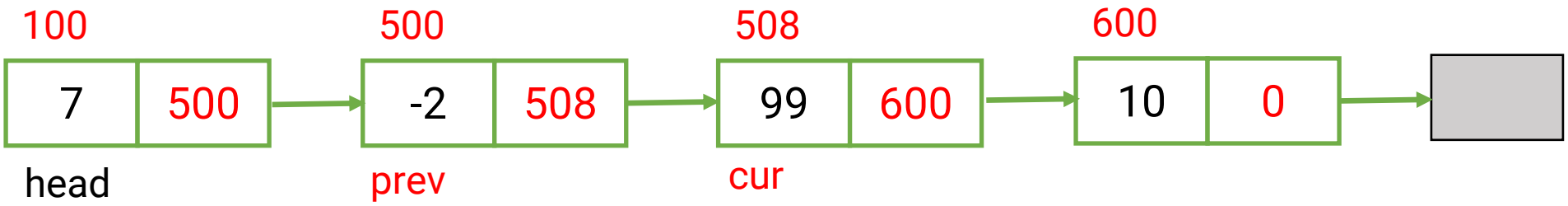
Minh họa removeMid(99)



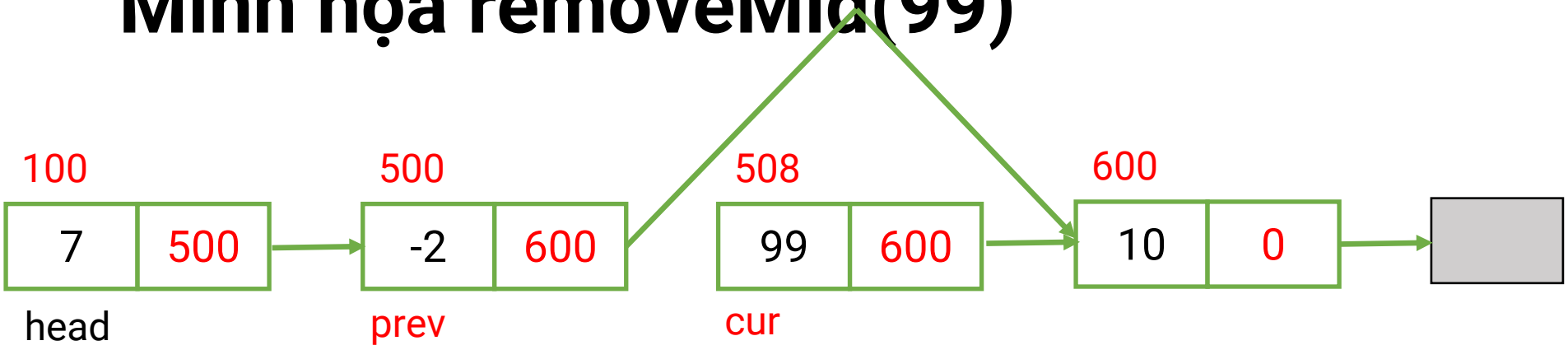
Minh họa removeMid(99)



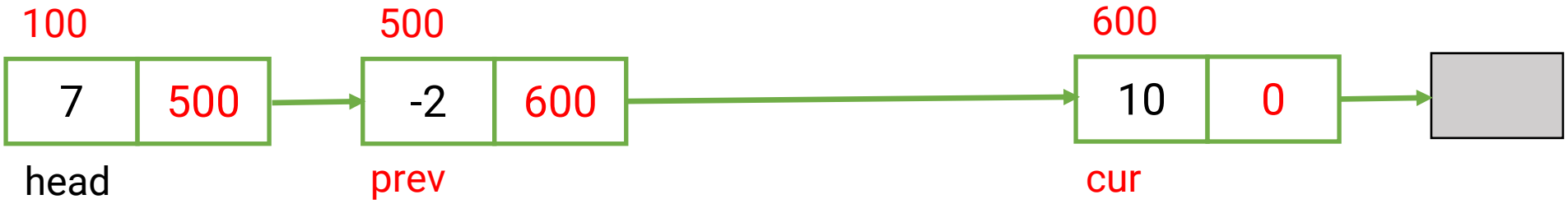
Minh họa removeMid(99)



Minh họa removeMid(99)



Minh họa removeMid(99)



Hỏi đáp