Data Structures

Linked List Traversal

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Singly Linked List: traversal

- **Traversal** Terminology: Walk through the elements of a data structure.
- **Print function** is a traversal function as it iterates oner the elements
- Traversal typically **O(n)** time for a complete iteration
- We usually code the traversal **iteratively**, but we can do it **recursively**
- Many similar traversal problems exists
    - Examples: min, max, nodes sum, is sorted, search for an element, get_nth

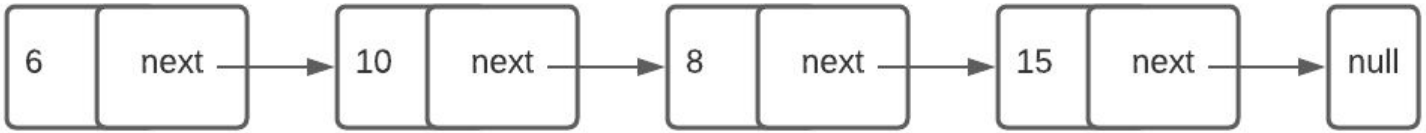
Print again (using for loops)

- Sometimes, it is more convenient to just iterate with for-loop
- Same code, but rearrange as a for loop
- *Common mistake*: corrupt your head node

```
void print() {
    // DON'T change head itself
    for (Node* cur = head; cur; cur = cur->next)
        cout << cur->data << " ";
    cout << "\n";
}
```

Get_nth item

- Let’s implement: Node* get_nth(int n)
  - The function get the nth node. If not found, it returns NULL
  - Below: get_nth(4) return Node with value 15
- Take 5-10 minutes to code it



Get_nth item

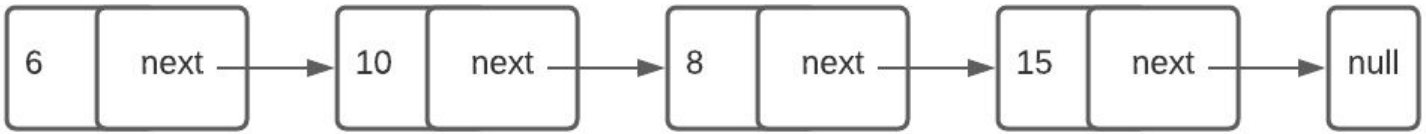
- You might code in several ways, but here is one clean way
- Just iterate and count up to N
- If the list ended without matching count, then it doesn't exist

```
Node* get_nth(int n) {
    int cnt = 0;
    for (Node* cur = head; cur; cur = cur->next)
        if (++cnt == n)
            return cur;

    return nullptr; // still more steps needed - NOT found
}
```

Search item

- Similarly, we can return 0-based index of a node with specific value
- Here: list.search(15) ⇒ 3
  list.search(99) ⇒ -1
- Give a trial



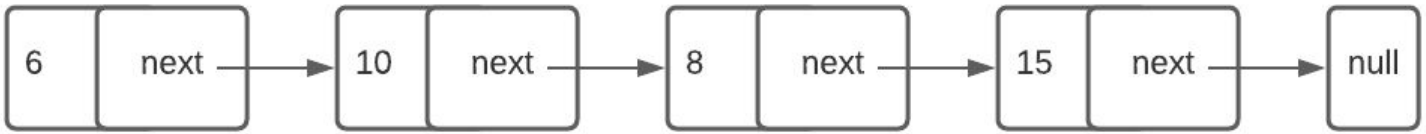
Search item

- Similarly, we can return 0-based index of a node with specific value
- Here: list.search(15) ⇒ 3
  list.search(99) ⇒ -1

```
int search(int value) {
    int idx = 0;
    for (Node* cur = head; cur; cur = cur->next, idx++)
        if (cur->data == value) // Common Mistake to use head
            return idx;
    return -1;
}
```

Improved Search

- Everytime we find the element, we shift it one step **left**
- For example searching for 15 will change the list to: 6 10 15 8
- Give a 10 min trial



Improved Search

- The main trick is to keep the previous. This is a **common** trick

```
int search_improved(int value) {
    int idx = 0;
    Node *previous = nullptr;   // let's keep pointer of previous
    for (Node* cur = head; cur; cur = cur->next, idx++) {
        if (cur->data == value) {
            if (!previous)
                return idx;
            swap(previous->data, cur->data);
            return idx - 1;
        }
        previous = cur;
    }
    return -1;
}
```

Improved Search: Common rewrite

- Common mistake:
  - Node *cur = head, prv = nullptr; [should be *prv]
  - cur = cur->next, prv = cur ⇒ prv = cur, cur = cur->next [observe the order]

```
int search_improved_v2(int value) {
    int idx = 0;

    for (Node *cur = head, *prv = nullptr; cur; prv = cur, cur = cur->next) {
        if (cur->data == value) {
            if (!prv)
                return idx;
            swap(prv->data, cur->data);
            return idx - 1;
        }
        ++idx;
    }
    return -1;
}
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."