Singly Linked List

Course: Introduction to Programming and Data Structures

Laltu Sardar

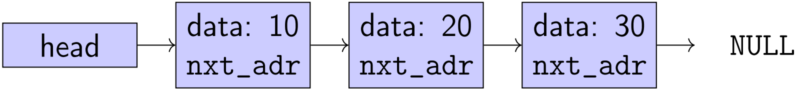
Institute for Advancing Intelligence (IAI),
TCG Centres for Research and Education in Science and Technology (TCG Crest)

Inventing Harmonious Future

1. Singly Linked List
    - Creating
    - Traversing
    - Inserting
    - Deleting
    - Some Linked List Problems

Singly Linked List

What is a Singly Linked List?



- A Singly Linked List is a data structure consisting of a sequence of elements, where each element points to the next one in the sequence.
- It is a linear data structure, similar to an array, but unlike arrays, linked lists do not have a fixed size.
- The main components of a singly linked list are nodes.

Structure of a Node

- Each node in a singly linked list consists of:
  - **Data**: The value stored in the node.
  - **Pointer**: A reference to the next node in the list.

C Structure Definition

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

Creating a Singly Linked List

- The head of the list is the first node.
- The next pointer of the last node is set to NULL, indicating the end of the list.

C Code Example

```c
struct Node* head = NULL;

struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = 10;
newNode->next = head;
head = newNode;
```

Traversing the List

- To traverse the list, we start from the head node and follow each next pointer until we reach NULL.

C Code Example

```c
struct Node* current = head;

while (current != NULL) {
    printf("%d ", current->data);
    current = current->next;
}
```

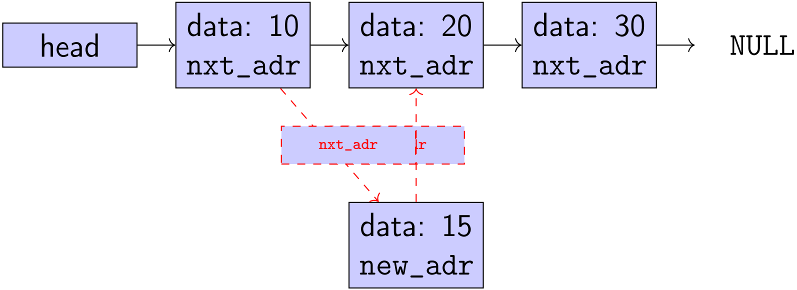
Inserting a Node

- Nodes can be inserted at the beginning, middle, or end of the list.

C Code Example

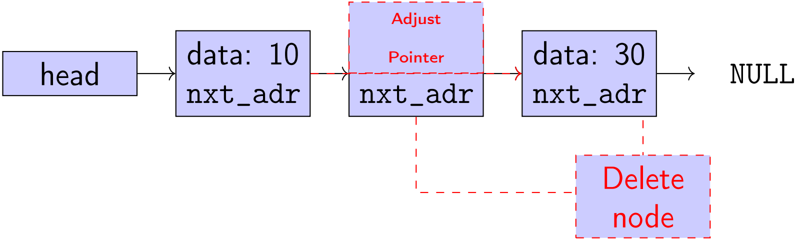
```c
void insertAtBeginning(struct Node** head, int newData) {
    struct Node* newNode=(struct Node*)malloc(sizeof(struct Node));
    newNode->data = newData;
    newNode->next = *head;
    *head = newNode;
}
```

Inserting a Node in the Middle



Deleting a Node

- To delete a node, adjust the next pointer of the previous node to skip the deleted node.



Deleting a Node

C Code Example

```c
void deleteNode(struct Node** head, int key) {
    struct Node* temp = *head;
    struct Node* prev = NULL;
    if (temp != NULL && temp->data == key) {
        *head = temp->next;
        free(temp);
        return;
    }
    while (temp != NULL && temp->data != key) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) return;
    prev->next = temp->next;
    free(temp);
}
```

Advantages of Linked Lists

- Dynamic size: Can grow or shrink as needed.
- Efficient insertions and deletions: No need to shift elements as in an array.
- Better use of memory: No need for pre-allocation.

Disadvantages of Linked Lists

- No random access: Must traverse the list to access elements.
- Extra memory space for the pointer: Each node requires additional space for a pointer.
- Less cache-friendly: Elements are not stored contiguously in memory.

Conclusion

- Singly Linked Lists are a fundamental data structure used in many applications.
- They offer flexibility with dynamic memory usage and efficient insertions/deletions.
- However, they come with trade-offs like no random access and additional memory overhead.

Important Operations on Singly Linked Lists I

- **Insertion**:
    - **At the Beginning**: Inserting a new node at the start of the list.
    - **At the End**: Inserting a new node at the end of the list.
    - **At a Specific Position**: Inserting a new node after a given node.
- **Deletion**:
    - **From the Beginning**: Removing the first node of the list.
    - **From the End**: Removing the last node (requires traversal to the last node).
    - **From a Specific Position**: Removing a node located after a specific node.
- **Traversal**:
    - **Forward Traversal**: Accessing each node of the list from the head to the last node.

Important Operations on Singly Linked Lists II

- **Search**:
  - **Search by Value**: Finding the first node containing a specific value.
  - **Search by Position**: Accessing the node at a particular index in the list.
- **Updating**:
  - Modifying the data stored in a specific node without altering the structure of the list.
- **List Reversal**:
  - Reversing the order of nodes in the list so that the first node becomes the last and vice versa.
- **Splitting**:
  - Dividing the list into two smaller lists at a given position.

Important Operations on Singly Linked Lists III

- **Concatenation**:
  - Merging two singly linked lists into a single list.
- **Length Calculation**:
  - Counting the number of nodes present in the list.

Some Linked List Problems

Problem 1: Remove Duplicates from Sorted List

Problem: Given the head of a sorted linked list, remove all duplicates such that each element appears only once.
Example:

- Input: $1 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 3$
- Output: $1 \rightarrow 2 \rightarrow 3$

Problem 2: Critical Points in a Linked List

Problem: Identify critical points in a linked list. A critical point is defined as a node where the value is either strictly greater than both neighbors or strictly less.

Example:

- Input: $1 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 5$
- Output: Critical points at 3 and 2

Problem 3: Cycle in a Linked List

Problem: Determine if a linked list has a cycle in it.
Example:

- Input: 3 → 2 → 0 → -4 (cycle to node 2)
- Output: True (Cycle exists)

Problem 4: Find Middle Element of Linked List

Problem: Given a singly linked list, return the middle node of the list.

Example:

- Input: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$
- Output: 3

Problem 5: Count Loop Length in Linked List

Problem: Find the length of the loop in a linked list if it exists.
Example:

- Input: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 2$ (cycle to node 2)
- Output: 3 (Length of loop is 3)

Problem 6: Sort a linked list

Problem: Sort a linked list using any sorting algorithm (E.g., merge sort, quick sort, etc.).

Example:

- Input: $4 \rightarrow 2 \rightarrow 1 \rightarrow 3$
- Output: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$

Problem 7: Separate Even and Odd values in a linked list

Problem: Separate the even and odd values of the linked list, maintaining their relative order.
Example:

- Input: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$
- Output: $2 \rightarrow 4 \rightarrow 1 \rightarrow 3$

Problem 8: Find Intersection of two linked lists

Problem: Find the node where two singly linked lists intersect.
Example:

- List 1: $1 \to 9 \to 1 \to 2 \to 4$
- List 2: $3 \to 2 \to 4$
- Output: Node with value 2

Problem 9: Rotate the List

Problem: Rotate the linked list to the right by k places.
Example:

- Input: $1 \to 2 \to 3 \to 4 \to 5$, $k = 2$
- Output: $4 \to 5 \to 1 \to 2 \to 3$

Problem 10: Reverse The Linked List

Problem: Reverse the entire linked list.
Example:

- Input: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$
- Output: $5 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$

Problem 11: Reverse the Segment

Problem: Reverse a portion of the linked list from position m to n.
Example:

- Input: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$, m = 2, n = 4
- Output: $1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 5$

Problem 12: Find Next Smaller value in Linked List

Problem: For each node, find the next node in the linked list with a smaller value.

Example:

- Input: 5 → 3 → 8 → 2
- Output: 3 → 2 → 2 → -1

Problem 13: Flatten a Linked List

Problem: Flatten a linked list where each node contains a pointer to another linked list.

Example:

- Input: $1 \rightarrow 2 \rightarrow 3$, and 1 points to list: $4 \rightarrow 5$
- Output: $1 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 3$

Problem 14: Reverse m size groups

Problem: Reverse the nodes in a linked list in groups of size m.
Example:

- Input: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$, $m = 3$
- Output: $3 \rightarrow 2 \rightarrow 1 \rightarrow 4 \rightarrow 5$

Thank You

for your attention.

Questions?

tcg crest

Inventing Harmonious Future

Laltu Sardar

laltu {dot} sardar [at]tcgcrest(.)org

https://laltu-sardar.github.io