Linked Lists: a generic node-based container

CMPT 145

Recap: The Node ADT

- Purpose:
  - Store data sequences.
- Implementations:
  - Dictionary with 2 values:
    1. A data value
    2. A reference to another node (or `None`)
- Operations:
  - Create a node
  - Set the data value for a given node
  - Set the reference to the next node for a given node
  - Return the data value of a given node
  - Return the reference to the node of a given node

The Linked List ADT

- Purpose:
    - Store data sequences.
- Implementation:
    - Dictionary containing a Node chain
- Operations:
    - Create a LList
    - Add to the list
    - Remove from the list
    - Search for a data value in the list
    - Access and change a data value in the list
    - Extend a list with another list
    - …

The Linked List Data Structure

The Linked List Data Structure is very much like the node-based Queue ADT. A linked list is a dictionary with the following keys:

size This keeps track of how many values are in the list.

head This is a reference to the first node in the node chain. An empty Linked List has no node chain, which we represent with None.

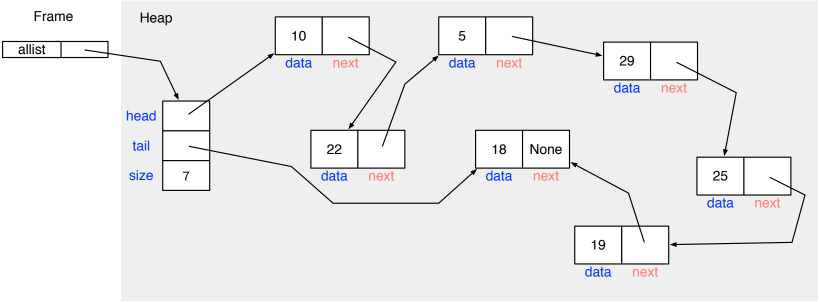
tail This is a reference to the last node in the chain. If the list is empty, this is None.

Implementing `create()`

The Linked List Data Structure is very much like the node-based Queue ADT. The `create()` operation is as follows:

```python
def create():
    """
    Purpose
        creates an empty list
    Return
        :return an empty list
    """
    llist = {}
    llist[’size’] = 0     # how many elements in the stack
    llist[’head’] = None  # node chain starts here
    llist[’tail’] = None  # and ends here; initially empty
    return llist
```

A Non-empty List on the Heap



Linked-list operations

create()

- Creates an empty list.

is_empty(alist)

- Checks if the given list has no data in it

size(alist)

- Returns the number of data values in the given list

Linked-list operations: Adding to either end

add_to_front(alist, val)

- Insert the given value `val` into the given Linked List `alist` so that the new value is at the front of the sequence of values.

add_to_back(alist, val)

- Add the given value `val` to the given Linked List `alist` so that the new value is at the end of the sequence of values.

Linked-list operations: removing from either end

remove_from_front(alist)

- Removes and returns the first value in the given Linked List `alist`.

remove_from_back(alist)

- Removes and returns the last value in the given Linked List `alist`.

Linked-list operations: basic indexing

get_data_at_index(alist, idx)

- Return the value stored in `alist` at the index `idx`.
- This function does not change the sequence; it simply reports what the value is stored at the given index.

set_data_at_index(alist, idx, val)

- Store `val` into `alist` at the index `idx`.
- This operation does not change the structure of the list. It simply replaces the value currently stored at `idx` with the given value.

Linked-list operations: search

value_is_in(alist, val)

- Check if the given value `val` is in the given list `alist`.
- Returns `True` if `val` is anywhere in the sequence, and `False` otherwise.

get_index_of_value(alist, val)

- Report the index of the given value `val` in the given list `alist`.
- If `val` appears more than once, the index of the first occurrence is reported.
- This function returns the tuple `(True, i)` if the given value appears in the list, where i is the index. If the value is not in the list, this function returns `False, None`.

Linked-list operations: structure changes

insert_value_at_index(alist, val, idx)

- Insert `val` into `alist` at index `idx`.
- This operation changes the structure of the list by adding a new value into the sequence, provided that `idx` is a valid index.
- Assume the index is non-negative, and in the range 0 to n, where n is the length of the list.
- If the index given is equal to the size of the list, the new value is added to the end of the sequence.

Linked-list operations: structure changes

delete_item_at_index(alist, idx)

- Delete the value at index `idx` in the given list `alist`.
- This operation changes the structure of the list, by removing a value.
- Assume that a valid index is non-negative, and in the range 0 to $n-1$, where n is the length of the list.

Linked-list operations: structure changes

delete_value(alist, val)

- Delete the value `val` from the given list `alist`.
- This operation changes the structure of the list, by removing a value.
- If the given value appears more than once, on the first occurrence is removed.
- If the given value does not appear in the sequence, the list remains unchanged.
- The function returns `True` is a value was deleted, or `False` if not.

Special cases for Linked List Operations

- Empty list
- List of one element
- Beginning of the list
- End of the list
- Index out of range

The Linked List ADT

- Linked lists were probably the first advanced data structure invented.
- Some languages provide linked lists as part of the base language.
- Once you have an ADT for a list data structure, you can vary the implementation to whatever is known to be best.
- Python's lists are not linked lists. They are something even more clever.