Justin Ciocoi

Oct. 5, 2023

CSCI 373 Class Notes

Advanced Data Structures and Algorithms

Singly Linked Lists

- Data is divided into nodes, which each contain *one* element
- Each node contains its own internal data as well as a link or pointer to the next node of the list
- Moving through the list using the sequential next pointers is called *pointer-hopping*
- Two nodes in a singly linked lists have special classifications
    - The *head* at the front of the list which is not pointed to by anything
    - The *tail* at the end of the list which points to a NULL value
- The structure is known as a singly linked list because each node stores a single link
- They can be maintained in an order and, unlike arrays in C++, do not have a predetermined size
- **Interface of StringNode Class**
    - ```
      class StringNode
      {
          private:
              string elem;
              StringNode* next;

              friend class StringLinkedList;
      };
      ```
- **Interface of StringLinkedList Class**
      ```

```
class StringLinkedList
{
    public:
        StringLinkedList();
        ~StringLinkedList();
        bool empty() const;
        const string& front() const;
        void addFront(const sstring& e);
        void removeFront();

    private:
        StringNode* head;
};
```

- **Insertion into a Singly Linked List**
  - You can insert at the head by:
    - Making a new node
    - Storing the address of the head in the next pointer of the new node
    - Setting the head variable to the new node
    - Below is the code representation of the above insertion algorithm
      - ```
        StringNode* v = new StringNode;

        v->next = head;

        head=v;
        ```

```
    ```

    ```
```

- You can insert at the end by:
  - Navigating the list to reach the last node
  - Create a new node
  - Attach the new node to the back of the last node
```

    - Ensure the new node points to NULL, which makes it the tail
    - Below is the code representation of the above insertion algorithm

        ```
        v=head;
        while(v->next != NULL)
            v=v->next;

        Node *n = new Node();

        v->next = n;

        n->next = NULL
        ```

- You can insert in the middle by:
    - Assuming we have a pointer to node `v`
    - We want to insert a node between `v` and `w` where `w` is a node immediately after `v`
    - First, create a new node, `z`
    - Copy the pointer to `w` to `z`
    - Make the pointer of `v` point to `z`
    - Below is the code representation of the above insertion algorithm

        ```
        Node *z = new Node();
        z->next = v->next;
        v->next = z;
        ```