Justin Ciocoi

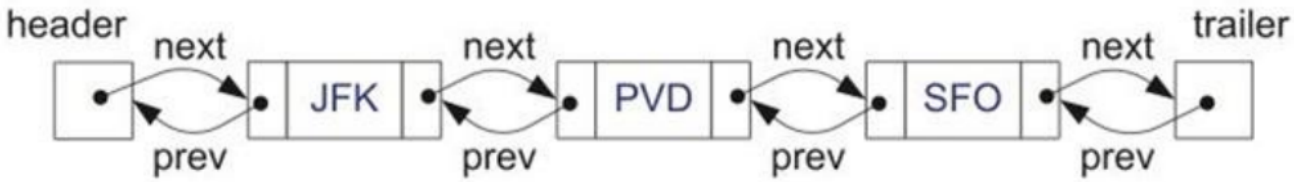
Oct. 5, 2023

CSCI 373 Class Notes

Advanced Data Structures and Algorithms

Doubly Linked Lists

- In a singly linked list, it takes long to remove any node other than the head
- To alleviate this issue, in a doubly linked list, pointers go in both directions
    - Each node stores two pointers, *next* and *prev*
- The *header* and *trailer* can be dummy nodes with pointers to the head and tail to provide quick access to these nodes
- The following is an example of a doubly linked list



Insertion in a Doubly Linked List

- Let us assume that we have a node, `v`, in a doubly linked list
    - `v` is not the trailer of the doubly linked list
- Now, we assume that `z` is a new node that we want to insert immediately after `v`
- Currently, we will assume `w` is the node immediately following `v`
- To insert a new element, we must first give it next and previous pointers to the appropriate elements and then re-define the next pointer for `v` and prev pointer for `w`
- Below is the code representation for this

```
z->prev=v;
z->next=w;
w->prev=z;
v->next=z;
```

- To remove, simply delete the node `z` and re-define the pointers between `v` and `w`

Interface of a Node in Doubly Linked List

```
typedef string Elem;
class dNode
{
    private:
        Elem elem;
        dNode* prev;
        dNode* next;
        friend class dLinkedList;
};
```

Interface of Doubly Linked List

```
class dLinkedList
{
    public:
        dLinkedList();
        ~dLinkedList();
        bool empty() const;
        const Elem& front() const;
        const Elem& back() const;
        void addFront(const Elem& e);
        void addBack(const Elem& e);
        void removeFront();
        void removeBack();
    private:
        dNode* header;
        dNode* trailer;
    protected:
        void add(dNode* v, const Elem& e);
        void remove(dNode* v);
}
```