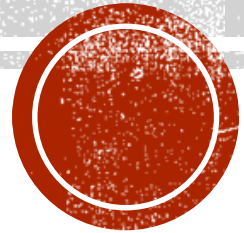
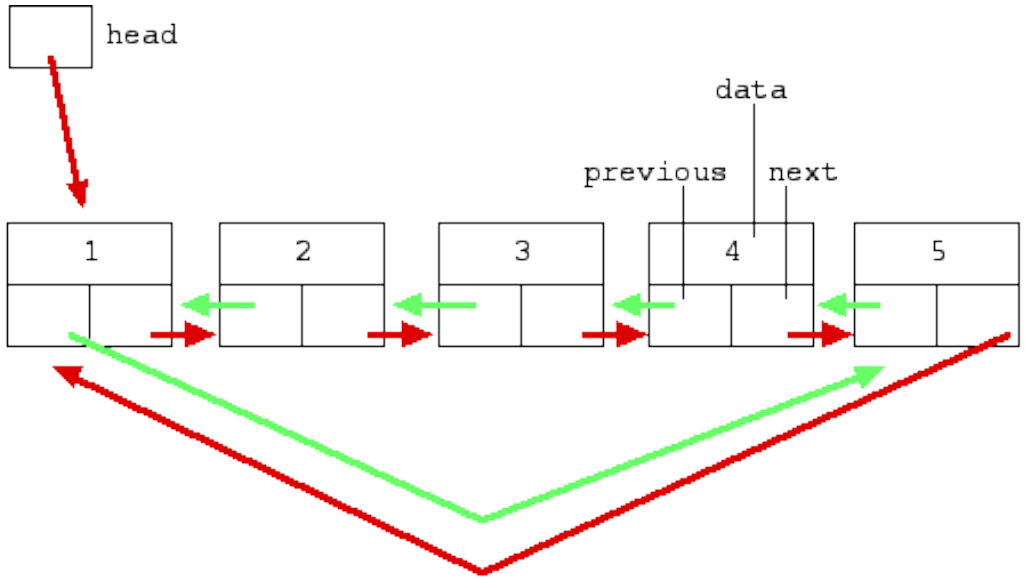
DOUBLY LINKED LISTS



DOUBLY LINKED LISTS

- Each node points to not only successor but the predecessor.
- There are two `NULL`: at the first and last nodes in the list.
- Given a node, it is easy to visit its predecessor.
- Convenient to traverse lists backwards.

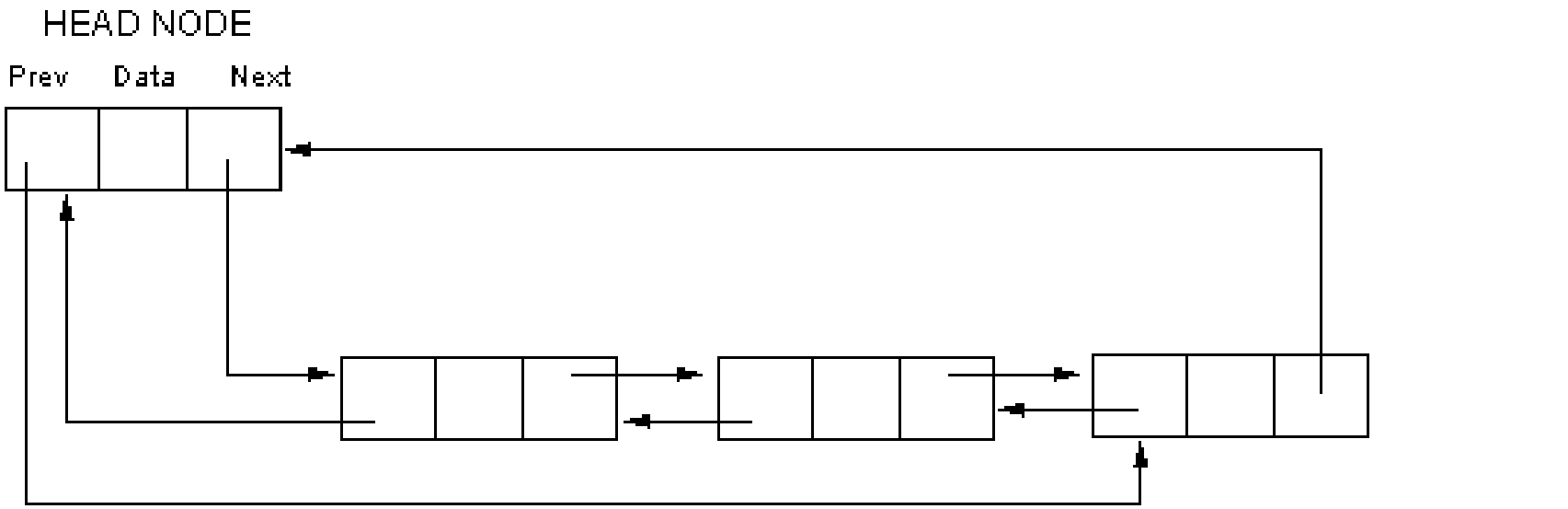


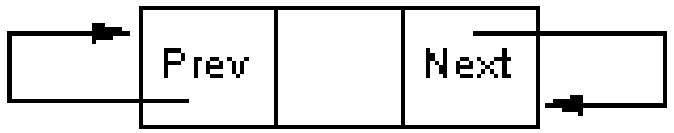
DOUBLY LINKED LISTS

```
typedef struct list_node *node_pointer;
typedef struct node{
    element item;
    node_pointer llink;
    node_pointer rlink;
  };
```

```
ptr = ptr->llink->rlink = ptr->rlink->llink
```

DOUBLE CIRCULAR LINKED LIST WITH HEAD NODE





INSERTION

```
void dinsert(node_pointer node, node_pointer new_node)
{
        /*insert newnode to the right of node*/
        newnode->llink = node;
        newnode->rlink = node->rlink;
        node->rlink->llink = newnode;
        node->rlink = newnode;
}
```

DELETION

```
void ddelete(node_pointer node, node_pointer deleted)
{
        /*delete from the doubly linked list*/
        if(node == deleted)
                //deletion of head node not permitted!
        else{
                deleted->llink->rlink = deleted->rlink;
                deleted->rlink->llink = deleted->llink;
                free(deleted);
        }
}
```

RUNNING TIMES

- insertion at head or tail is in O(1)
- deletion at either end is on O(1)
- element access is still in O(n)

ALGORITHMS