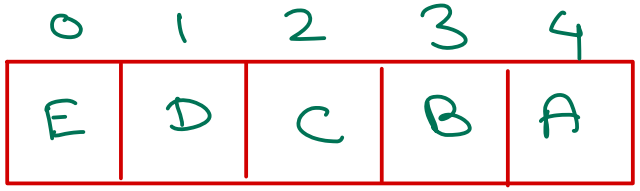
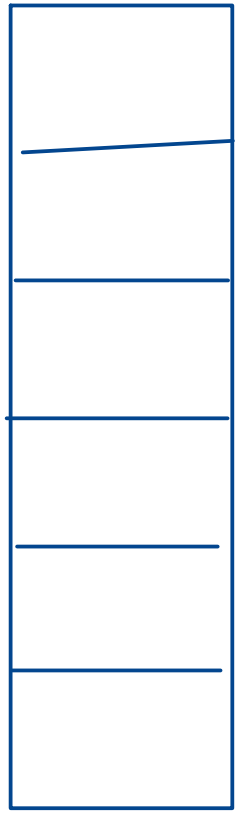
Data Structure & Algorithms

Nilesh Ghule

Stack / Queue – Competitive Programming

- Reverse array, string or linked list.



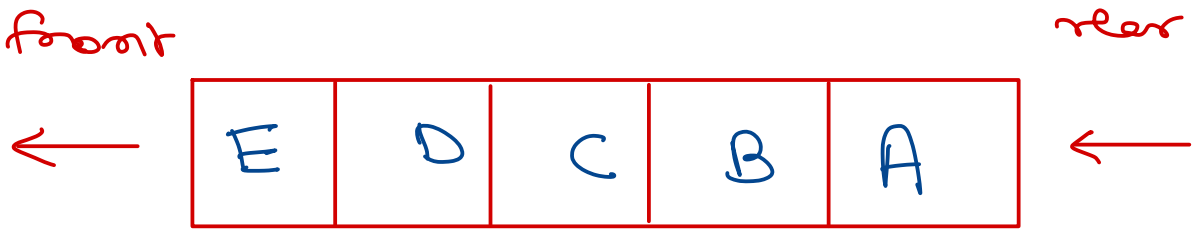



```
for(i=0;i<n;i++)
    s.push(arr[i]);

i=0;
while(!s.isempty()){
    arr[i]=s.pop()
    i++;
}
```

Stack / Queue – Competitive Programming

- Create stack using queue.

LIFO FIFO



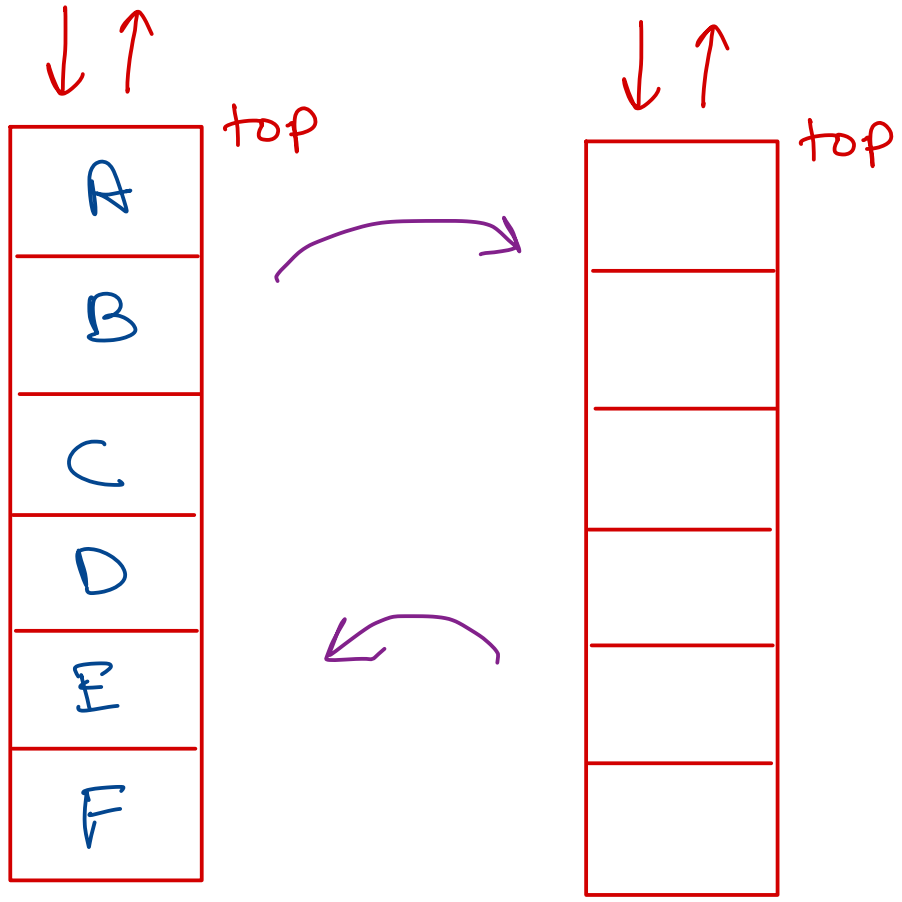

```
class myStack {
    Queue q;
    push(val) {
        size = q.size();
        q.push(val);
        for(i=1; i<=size; i++) {
            temp=q.pop();
            q.push(temp);
        }
    }
    pop() {
        return q.pop();
    }
}
```

Stack / Queue – Competitive Programming

- Create queue using stack? → Assignment

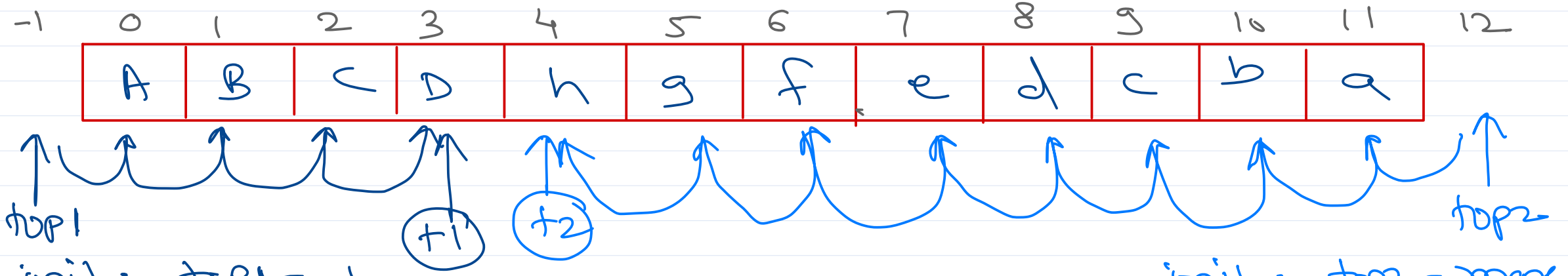
FIFO LIFO

```
class MyQueue {
    Stack s = new Stack();
    push(val) {
        Stack t = new stack();
        while(!s.isEmpty())
            t.push(s.pop());
        s.push(val);
        while(!t.isEmpty())
            s.push(t.pop());
    }
}
```



2-Stack

How to implement two stacks in single array efficiently?



init: top1 = -1

push: top1++;
arr[top1] = val;

pop:
top1--;

isempty:
top1 == -1

isFull:
top1 + 1 == top2

init: top2 = max

push: top2--;
arr[top2] = val;

pop: top2++;

isempty:
top2 == max

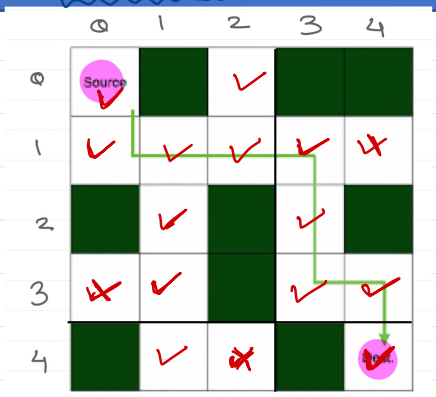
- Back tracking

Rat in a maze problem.

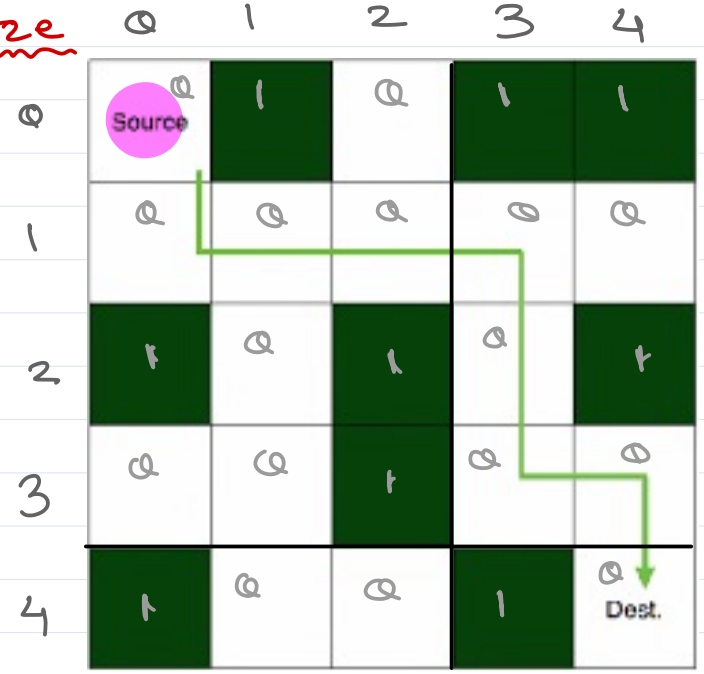
```
class cell {
    int r, c;
}

Stack<Cell> s = new Stack();
Cell src = (0,0), dest = (4,4);
s.push(src);
while( !s.isEmpty()) {
    cur = s.pop();
    if(cur == dest)
        return true;
    neighbours = find cur neighbours
    for( n : neighbours) {
        if ( n is not obstacle &&
             n is not visited){
            s.push(n);
            visited[n.row][n.col]=true;
    }
}
```

```
int maze[][] = {
    { 0, 1, 0, 1, 1 },
    { 0, 0, 0, 0, 0 },
    { 1, 0, 1, 0, 1 },
    ...
};
```

visited / not



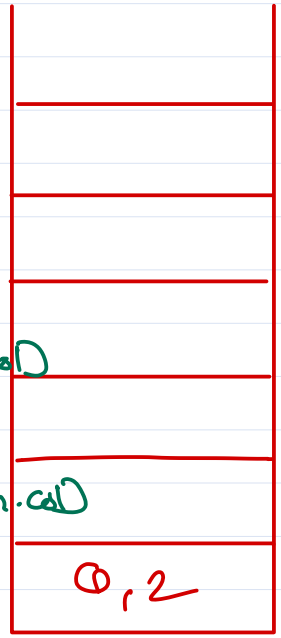
maze



r+1
r-1
c+1
c-1

maze[n.row][n.col] != 1

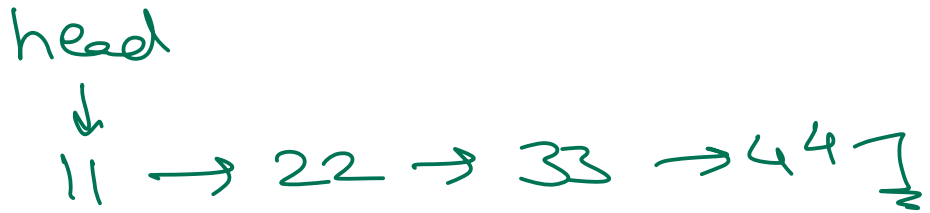
!visited[n.row][n.col]

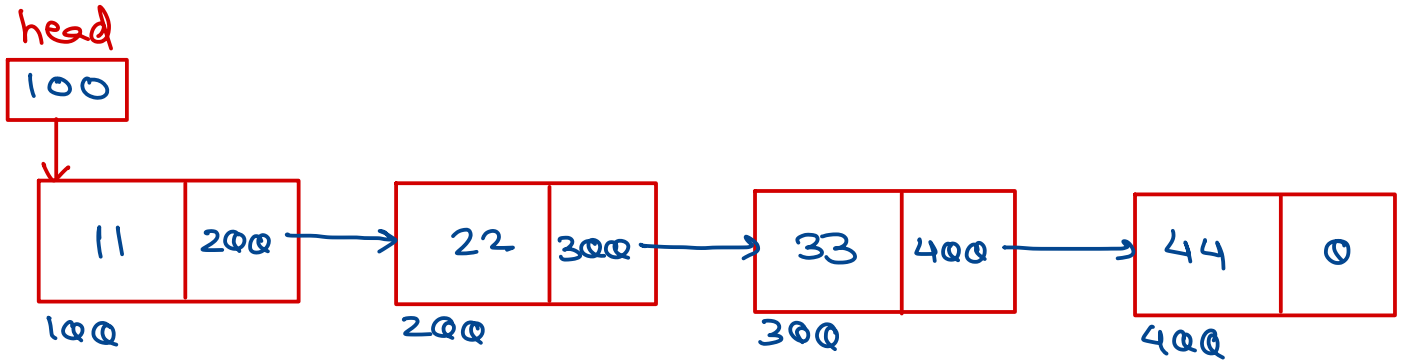


```
neighbours = {
        new Cell(cur.row+1, cur.col   ),
        new Cell (cur.row-1, cur.col  ),
        new Cell (cur.row, cur.col+1 ),
        new Cell (cur.row, cur.col-1 )
};
```

Linked List

- Linked List is list of items linked together.
- Each item in linked list is called as Node.
- Each node contains data and pointer/reference to the next node.
- Linked list is linear data structure.

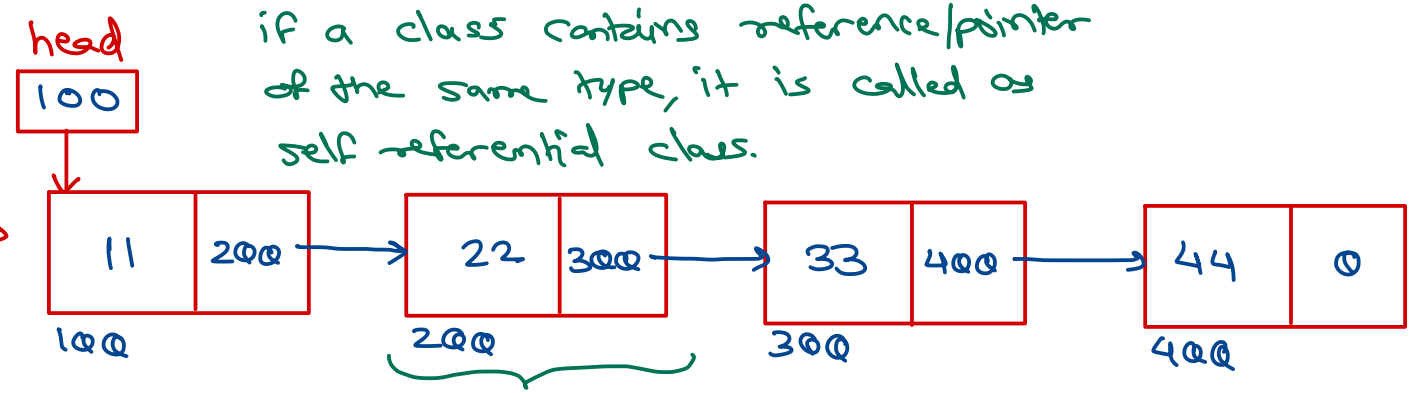




- Linked list ADT
  - addFirst()
  - addLast()
  - addAtPos()
  - deleteFirst()
  - deleteLast()
  - deleteAsPos()
  - deleteAll()
  - traverse()

Linked List

- There four types of linked list.
  - Singly linear linked list
  - Singly circular linked list
  - Doubly linear linked list
  - Doubly circular linked list




```
class Node {
    int data;
    Node *next;
    get/set
    friend class List;
};
```

```
class List {
    Node *head;
    addLast() { }
    addFirst() { }
    display() { }
    ⋮
};
```

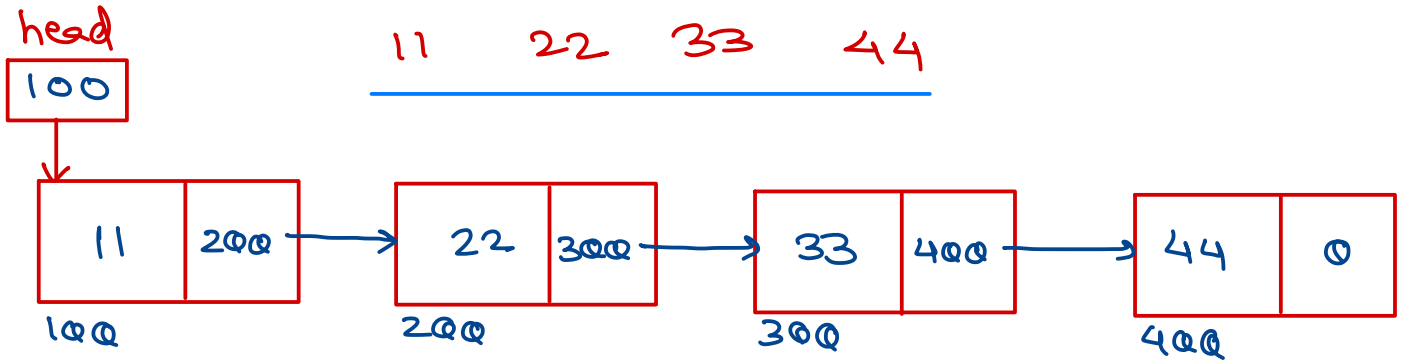
```
class Node {
pr  int data;
pr  Node next;
    get/set
};
```

Java →

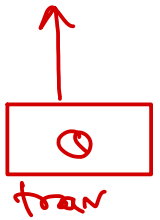
```
class List {
    Node head;
    addLast() { }
    addFirst() { }
    display() { }
    ⋮
};
```

static inner class

Singly Linear Linked List – display()

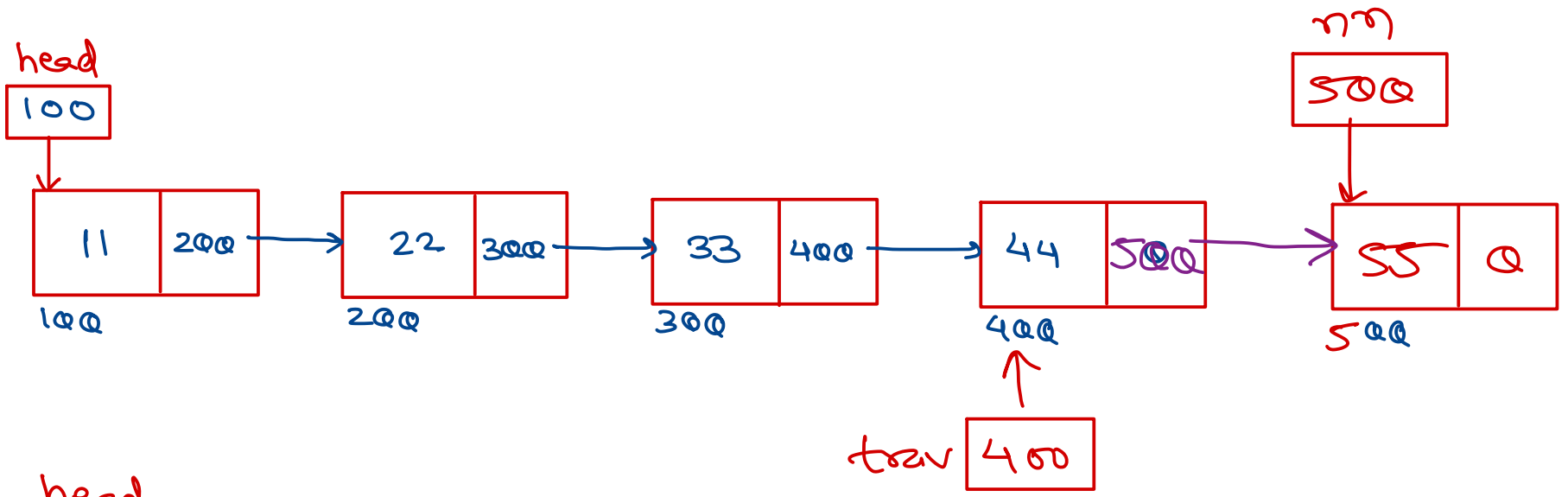


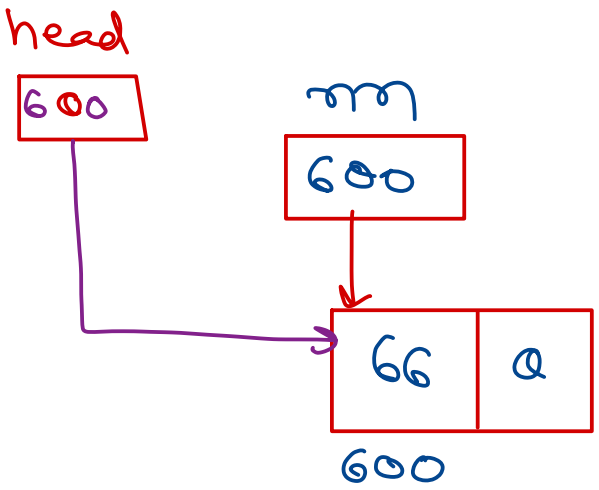
initially - list is empty.




```
trav = head;
while (trav != null) {
    print (trav.data);
    trav = trav.next;
}
```

Singly Linear Linked List - addLast()






```
nn = new Node(val);
if(head==null)
    head=nn;
else {
    trav = head;
    while(trav.next!=null)
        trav = trav.next;
    trav.next = nn;
}
```

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>