Data Structures and Algorithms

Agenda

- Sorting
- ADT
- Queue
- Stack
- Applications

Sorting

- Stable sort vs Unstable sort
- In-place sort vs Out-place sort

Stable sort vs Unstable sort

- Array: [{A, 65}, {B, 90}, {C, 55}, {D, 85}, {E, 55}, {F, 65}]
- Stable sort:
    - Equal elements maintains their relative order as in original array -- Guaranteed.
    - [{C, 55}, {E, 55}, {A, 65}, {F, 65}, {D, 85}, {B, 90}]
    - e.g. Bubble, Insertion, ...
- UnStable sort:
    - Equal elements may not maintain their relative order as in original array.
    - [{C, 55}, {E, 55}, {F, 65}, {A, 65}, {D, 85}, {B, 90}]
    - e.g. Selection.

In-place sort vs Out-place sort

- In-place sort

    - No additional space requires for holding array element.
    - Aux Space complexity is O(1)
    - e.g. Selection, Bubble, Insertion, ...
- Out-place sort
    - Additional space requires for holding sorted array element.
    - Aux Space complexity is O(n) -- without stack space.
    - e.g. Merge.

ADT - Abstract Data Type

- Data structure
    - How data is organized in memory?
    - How operations are performed on that data?
- From user perspective, all data structures are ADTs.

(Static) Array ADT

- Operations
    - Write element on given index: arr[i] = x
    - Read element from given index: y = arr[i]
- Advanced Operations
    - Sorting
    - Searching
    - Traversing

Queue ADT

- FIFO behaviour
- Operations
    - Insert in Queue -- Push()
    - Delete from Queues -- Pop()
    - Read next Element -- Peek()

    - isEmpty() -- True/False
    - isFull() -- True/False
- isFull() operation is applicable if storage capcity is fixed.

Stack ADT

- LIFO behaviour
- Operations
    - Insert -- Push()
    - Delete -- Pop()
    - Read next -- Peek()
    - isEmpty() -- True/False
    - isFull() -- True/False
- isFull() operation is applicable if storage capcity is fixed.

Stack vs Queue

- S: LIFO
- Q: FIFO
- S: Push and Pop are done from same end -- "top"
- Q. Push and Pop are done from different ends -- "rear" and "front"

Queue Applications

- Operating Systems
    - Message queue (IPC)
    - Waiting queue (IO waiting)
    - Ready queue (CPU scheduling e.g. FCFS, SJF, ...)
- Breadth First Search (Tree and Graph)

Stack Applications

- Process stack -- Function call
- Depth First Search (Tree and Graph)
- Back-tracking
- Math expression solve
- Parenthesis balancing

Circular Queue

```
class CircularQueue {
    private int[] arr;
    private int front, rear;
    private int count;

    public CircularQueue(int size) {
        arr = new int[size];
        front = -1;
        rear = -1;
        count = 0;
    }
    public void push(int val) {
        rear = (rear+1) % arr.length;
        arr[rear] = val;
        count++;
    }
    public int pop() {
        front = (front+1) % arr.length;
        count--;
        return arr[front];
    }
    public int peek() {
        int index = (front+1) % arr.length;
        return arr[index];
    }
```

```
    public boolean isEmpty() {
        return (count == 0);
    }
    public boolean isFull() {
        return (count == arr.length);
    }
}
```

Data structure Implementations

User defined implementations

- Stack -- LIFO
- Linear Queue -- FIFO
- Circular Queue -- FIFO

Programming Languages - Built-in Libraries

Java -- Collection Framework

- Package: java.util
    - Interfaces: Collection, Set, List, Queue, Map
    - Classes: ArrayList, LinkedList, HashSet, HashMap, ...
    - Helper classes: Collections, Arrays

Stack class

```
```Java
Stack<String> s = new Stack<String>();
s.push("A");
s.push("B");
```
```

```
s.push("C");
System.out.println("Topmost Element: " + s.peek()); // C
while(!s.isEmpty()) {
    System.out.println("Popped Element: " + s.pop());   // C, B, A
}
```
```

**Queue interface/LinkedList class**

- Queue<> operations:
  - To add element: offer()
  - To delete element: poll()
  - To get topmost element: peek()

```
//Queue<String> q = new LinkedList<String>();
LinkedList<String> q = new LinkedList<String>();
q.offer("A");   // like push()
q.offer("B");   // like push()
q.offer("C");   // like push()
System.out.println("Topmost Element: " + q.peek()); // A
while(!q.isEmpty()) {
    System.out.println("Popped Element: " + q.poll()); // A,B,C
}
```

C++ -- STL

- Container classes: vector, list, stack, queue, map, multimap, set, ...

Python -- Collections
```

- Collections
  - List: `[ ... ]`
  - Dictionary: `{ ... }`
  - Tuple: `( ... )`

Expressions

- "2+3*4"
  - Operands -- ASCII value check
  - Operators -- Non-operands
- "23 + 354 * 5"
  - Tokens are separated by space.
  - String[] tokens = infix.split(" ");
  - Operands -- Integer.parseInt() if work
  - Operators -- Non-operands

Assignments

1. Implement circular queue using counter method.
2. Implement stack using array. Keep initial value of top as 0. Modify other operations accordingly.
3. Paperwork: Convert following examples from Infix to Prefix and Postfix.
   - K + L - M*N + (O^P) * W/U/V * T + Q
   - (A + B) * C - (D - E) * (F + G)
4. Paperwork: Convert followinf expression to Prefix using stack algorithm. Refer code.
   - 5+9-4*(8-6/2)+1$(7-3)
5. Create an array of integers. Reverse the array using stack.