23BCE2086

SHRIDHARAN VK

CC DOMAIN TASK SUBMISSION

LEVEL 0

DOCUMENTATION

Features Implemented:

- Node creation
- Insertion at head and tail
- Forward and backward traversal

Technologies Used:

- Java
- VS CODE

Setup Instructions:

1. Compile using javac CC0.java
2. Run using java CC0

Implementation Details:

- insertHead() and insertTail() maintain head and tail references.
- traverseForward() and traverseBackward() print elements in both directions.

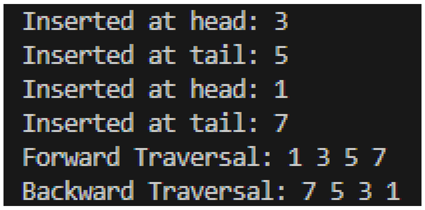
Time and Space Complexity:

- Insertion: **O(1)**
- Traversal: **O(n)**
- Space: **O(n)** (each node stores data and two pointers)

CODE

```
class DoublyLinkedList {
   class Node {
      int data;
      Node prev, next;
      Node(int d) { data = d; }
   }
   private Node head, tail;

   void insertAtHead(int data) {
      Node newNode = new Node(data);
      if (head == null) {
         head = tail = newNode;
      } else {
         newNode.next = head;
         head.prev = newNode;
         head = newNode;
      }
      System.out.println("Inserted at head: " + data);
   }

   void insertAtTail(int data) {
        Node newNode = new Node(data);
        if (tail == null) {
            head = tail = newNode;
        } else {
            tail.next = newNode;
            newNode.prev = tail;
            tail = newNode;
        }
        System.out.println("Inserted at tail: " + data);
    }

    void traverseForward() {
        Node temp = head;
        System.out.print("Forward Traversal: ");
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }

    void traverseBackward() {
        Node temp = tail;
        System.out.print("Backward Traversal: ");
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.prev;
        }
        System.out.println();
    }
}

public class CC0 {
    public static void main(String[] args) {
        DoublyLinkedList dll = new DoublyLinkedList();
        dll.insertAtHead(3);
        dll.insertAtTail(5);
        dll.insertAtHead(1);
        dll.insertAtTail(7);
        dll.traverseForward();
        dll.traverseBackward();
    }
}
```

OUTPUT



```
Inserted at head: 3
Inserted at tail: 5
Inserted at head: 1
Inserted at tail: 7
Forward Traversal: 1 3 5 7
Backward Traversal: 7 5 3 1
```

LEVEL 1

DOCUMENTATION

Stack with Min/Max Operations

Technologies Used:

- Java
- VS CODE

Features Implemented:

- Push, pop, top retrieval in **O(1)**
- Get min/max in **O(1)**

Setup:

1. Compile using javac CC1.java
2. Run using java CC1

Time Complexity:

- push(x): **O(1)**
- pop(): **O(1)**
- top(): **O(1)**
- getMin(): **O(1)**
- getMax(): **O(1)**

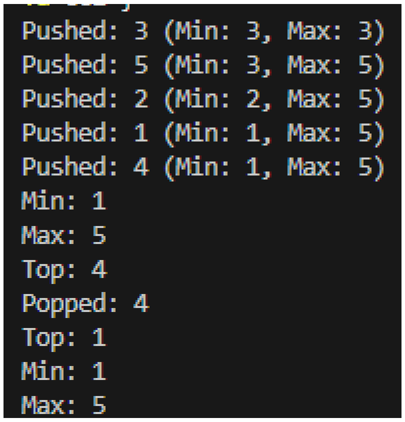
Space Complexity:

- **O(n)** (each node stores value, min, and max)

CODE

```
class MinMaxStack {
    class Node {
        int value, min, max;
        Node next;
        Node(int v, int mn, int mx, Node n) { value = v; min = mn; max = mx; next = n; }
    }
    private Node top;

    void push(int x) {
        if (top == null) {
            top = new Node(x, x, x, null);
            System.out.println("Pushed: " + x + " (Min: " + x + ", Max: " + x + ")");
        } else {
            top = new Node(x, Math.min(x, top.min), Math.max(x, top.max), top);
            System.out.println("Pushed: " + x + " (Min: " + top.min + ", Max: " + top.max + ")");
        }
    }
    void pop() {
        if (top != null) {
            System.out.println("Popped: " + top.value);
            top = top.next;
        } else {
            System.out.println("Stack is empty");
        }
    }

    int top() {
        if (top == null) {
            System.out.println("Top: Stack is empty");
            return -1;
        }
        System.out.println("Top: " + top.value);
        return top.value;
    }

    int getMin() {
        if (top == null) {
            System.out.println("Min: Stack is empty");
            return -1;
        }
        System.out.println("Min: " + top.min);
        return top.min;
    }

    int getMax() {
        if (top == null) {
            System.out.println("Max: Stack is empty");
            return -1;
        }
        System.out.println("Max: " + top.max);
        return top.max;
    }
}

public class StackOperations {
    public static void main(String[] args) {
        MinMaxStack stack = new MinMaxStack();
        stack.push(3);
        stack.push(5);
        stack.push(2);
        stack.push(1);
        stack.push(4);
        stack.getMin();

        stack.getMax();

        stack.top();

        stack.pop();

        stack.top();

        stack.getMin();

        stack.getMax();

    }

}
```

OUTPUT



```
Pushed: 3 (Min: 3, Max: 3)
Pushed: 5 (Min: 3, Max: 5)
Pushed: 2 (Min: 2, Max: 5)
Pushed: 1 (Min: 1, Max: 5)
Pushed: 4 (Min: 1, Max: 5)
Min: 1
Max: 5
Top: 4
Popped: 4
Top: 1
Min: 1
Max: 5
```

LEVEL 2

DOCUMENTATION

Features Implemented:

- Add intervals with automatic merging
- Maintain non-overlapping intervals

Technologies Used:

- Java

- VS CODE

Setup Instructions:

1. Compile using javac Level2.java
2. Run using java Level2

Implementation Details:

- addInterval(start, end): Merges overlapping intervals.
- getIntervals(): Prints stored intervals.

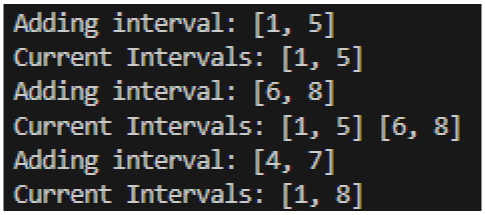
Time and Space Complexity:

- addInterval(): **O(n)** (scans existing intervals for merging)
- getIntervals(): **O(n)**
- Space: **O(n)** (stores non-overlapping intervals)

CODE

```
class IntervalMerger {
    class Interval {
        int start, end;
        Interval next;
        Interval(int s, int e) { start = s; end = e; }
    }
    private Interval head;

    void addInterval(int start, int end) {
        System.out.println("Adding interval: [" + start + ", " + end + "]");
        Interval newInterval = new Interval(start, end);
        if (head == null) {
            head = newInterval;
            return;
        }
        Interval prev = null, curr = head;
        while (curr != null && curr.end < start) {
            prev = curr;
            curr = curr.next;
        }
        while (curr != null && curr.start <= end) {
            start = Math.min(start, curr.start);
            end = Math.max(end, curr.end);
            curr = curr.next;
        }
        newInterval.start = start;
        newInterval.end = end;
        newInterval.next = curr;
        if (prev == null) head = newInterval;
        else prev.next = newInterval;
    }

    void getIntervals() {
        System.out.print("Current Intervals: ");
        Interval temp = head;
        while (temp != null) {
            System.out.print("[" + temp.start + ", " + temp.end + "] ");
            temp = temp.next;
        }
        System.out.println();
    }
}

public class CC2  {
    public static void main(String[] args) {
        IntervalMerger im = new IntervalMerger();
        im.addInterval(1, 5);
        im.getIntervals();
        im.addInterval(6, 8);
        im.getIntervals();
        im.addInterval(4, 7);
        im.getIntervals();
    }
}
```

OUTPUT



```
Adding interval: [1, 5]
Current Intervals: [1, 5]
Adding interval: [6, 8]
Current Intervals: [1, 5] [6, 8]
Adding interval: [4, 7]
Current Intervals: [1, 8]
```

LEVEL 3

DOCUMENTATION

Features Implemented:

- Key-value storage with automatic expiration.
- Fast retrieval of values if they haven't expired.
- Automatic cleanup of expired keys during retrieval or insertion.
- Simulated time progression to test expiration behavior.

Technologies Used:

- Java
- VS CODE

Setup Instructions:

1. Compile and run the CC3 class.
2. Use set(key, value, expiryTime) to store key-value pairs with an expiration time.
3. Use get(key) to retrieve values if they exist and haven't expired.
4. Use incrementTime(seconds) to simulate time progression and observe expiration behavior.

Operations and Complexity:

Operation	**Time Complexity**	**Space Complexity**
set(key, value, expiryTime)	O(1)	O(n) (Active keys)
get(key)	O(n) (worst case due to cleanup)	O(n)
cleanExpiredKeys()	O(n)	O(n)

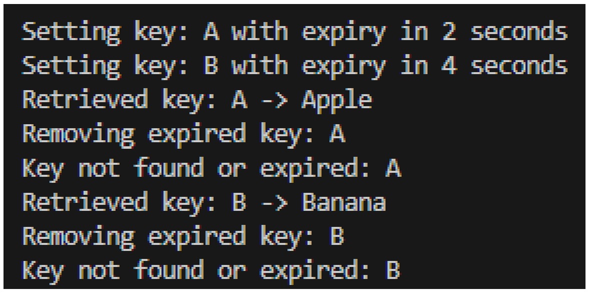
Execution Example:

1. **Set** "A" with expiry 2 seconds.
2. **Set** "B" with expiry 4 seconds.
3. **Retrieve** "A" before expiry → Returns value.
4. **Increment time** by 3 seconds.
5. **Try retrieving** "A" (should be expired) → Returns null.
6. **Retrieve** "B" before expiry → Returns value.
7. **Increment time** by 2 seconds.
8. **Try retrieving** "B" (should be expired) → Returns null.

CODE

```
class TimeBasedCache {
    class Node {
        String key;
        String value;
        long expiry;
        Node next;
        Node(String k, String v, long e, Node n) { key = k; value = v; expiry = e; next = n; }
    }
    private Node head;
    private long currentTime = 0;

    void set(String key, String value, long expiryTime) {
        cleanExpiredKeys();
        long expiry = currentTime + expiryTime;
        System.out.println("Setting key: " + key + " with expiry in " + expiryTime + " seconds");
        head = new Node(key, value, expiry, head);
    }

    String get(String key) {
        cleanExpiredKeys();
        Node temp = head;
        while (temp != null) {
            if (temp.key.equals(key)) {
                System.out.println("Retrieved key: " + key + " -> " + temp.value);
                return temp.value;
            }
            temp = temp.next;
        }
        System.out.println("Key not found or expired: " + key);
        return null;
    }

    void cleanExpiredKeys() {
        while (head != null && head.expiry < currentTime) {
            System.out.println("Removing expired key: " + head.key);
            head = head.next;
        }
        if (head == null) return;
        Node prev = head, temp = head.next;
        while (temp != null) {
            if (temp.expiry < currentTime) {
                System.out.println("Removing expired key: " + temp.key);
                prev.next = temp.next;
            } else {
                prev = temp;
            }
            temp = temp.next;
        }
    }
    void incrementTime(long seconds) {
        currentTime += seconds;
    }
}


public class CC3 {
    public static void main(String[] args) {
        TimeBasedCache cache = new TimeBasedCache();
        cache.set("A", "Apple", 2);
        cache.set("B", "Banana", 4);
        cache.get("A");
        cache.incrementTime(3);
        cache.get("A");
        cache.get("B");
        cache.incrementTime(2);
        cache.get("B");}
}
```

OUTPUT



```
Setting key: A with expiry in 2 seconds
Setting key: B with expiry in 4 seconds
Retrieved key: A -> Apple
Removing expired key: A
Key not found or expired: A
Retrieved key: B -> Banana
Removing expired key: B
Key not found or expired: B
```