CSC3100 Tutorial 6

Xingjian Wang 117010266@link.cuhk.edu.cn

Minstack

- Design a stack that supports push, pop, top, and retrieving the minimum element in constant time

- The class contains the following functions:
  Minstack(): initialize the stack object
  void push(int val): pushes the element “val” to the stack
  void pop(): removes the top element
  int top(): return the top element
  int getMin(): retrieves the minimum element of the stack

Minstack

- Since minstack is an FIFO data structure, pushing to the stack doesn’t change the data pushed before.
- Therefore, we can record the current minimum when each element is at the top of the stack. When that certain element becomes the top element again, the minimum element of the stack is still as recorded.
- Example: Consider a stack [5, 3, 1, 2, 4], the current top element is 5 and the minimum is 1. After we push two random elements and pop two elements, the stack remains to be the same and the minimum should also be 1.

```java
import javafx.util.Pair;
import java.util.*;

public class minstack {
    Stack<Pair<Integer, Integer>> stack = new Stack<Pair<Integer, Integer>>();
    public void push(int x) {
        // only push the old minimum value when the current
        // minimum value changes after pushing the new value x
        if(stack.empty()){
            Pair<Integer, Integer> element = new Pair<Integer, Integer>(x, x);
            stack.push(element);
        } else {
            Pair<Integer, Integer> element;
            Integer currentMin = stack.peek().getValue();
            if (x < currentMin) {
                element = new Pair<Integer, Integer>(x, x);
            } else {
                element = new Pair<Integer, Integer>(x, currentMin);
            }
            stack.push(element);
        }
    }

    public void pop() {
        stack.pop();
    }

    public int top() {
        return stack.peek().getKey();
    }

    public int getMin() {
        return stack.peek().getValue();
    }
}
```

Next greater element

- The next greater element of some element “x” in an array is the first greater element that is to the right of “x” of the same array. e.g. for array [3, 2, 1, 5, 4], the next greater element of 3 is 5.
- You are given an array “nums”. Return an array “ans” with length same as “nums”, where “ans[i]” contains the next greater element described above.
- Example: nums = [3, 1, 2, 5, 4], then ans = [5, 2, 5, -1, -1]

Next greater element

- The usual way takes O(n^2) time by iterating through the array twice
- However, we can use stack named “stack0” to achieve O(n) time.
- If there is a consecutive sequence of numbers, such that nums[k] >= nums[k+1] >= … >= nums[t], then none of the numbers nums[k+1], nums[k+2],…, nums[t] would be the next greater element of nums[k].
- Also, nums[n] is the next greater element of nums[k] only if nums[n] >= nums[k+1]

Next greater element

- When iterating through the array at iterate i, compare the nums[i] with stack.top(). If stack.top() > nums[i], push nums[i] to the stack. If stack.top() < nums[i], then nums[i] is the next largest element of stack.top(). We call stack.pop() and continue our comparing until there is no element or stack is empty.

- After iterating through the whole array, the elements still in the stack are the elements don’t have next greater element

- Minor issue: How to keep track of the elements’ location in the array? Answer: Use java Pair<Integer, Integer> type. Store the element in the key part and the location in the value part.

```java
import javafx.util.Pair;
import java.util.*;

public class nextgreater {
    public static void main(String[] args) {
        int nums[] = {3, 1, 2, 5, 4};
        int[] result = new int[5];
        int numsLength = 5;
        Stack<Pair<Integer, Integer>> stack0 = new Stack<Pair<Integer, Integer>>();

        Pair<Integer, Integer> p = new Pair<Integer, Integer>(nums[0], 0);
        stack0.push(p);
        for (int i = 1; i < numsLength; i++) {
            int currentElement = nums[i];
            while (!stack0.empty()) {
                Pair<Integer, Integer> top = stack0.peek();
                Integer topVal = top.getKey();
                Integer topLocation = top.getValue();
                if (currentElement > topVal) {
                    result[topLocation] = currentElement;
                    stack0.pop();
                } else {
                    break;
                }
            }
            Pair<Integer, Integer> p0 = new Pair<Integer, Integer>(nums[i], i);
            stack0.push(p0);
        }

        while (!stack0.empty()) {
            Pair<Integer, Integer> top = stack0.peek();
            result[top.getValue()] = -1;
            stack0.pop();
        }

        for (int i = 0; i < numsLength; i++) {
            System.out.println(result[i]);
        }
    }
}
```