CS1020E: Data Structures and Algorithms I

Tutorial 6 – Stack & Linked List

(Week starting 9 March 2015)

1. **Brackets.** [Classical problem]. Given a sequence of brackets which only contains '(', ')' ,'[', ']'. Check if the sequence is a legal, according to the rules below.
   1) [] is legal
   2) () is legal
   3) If sequence S is legal, then so are [S] and (S).
   4) If both sequence S and sequence T are legal, then so is ST.

   For example: ((([]))[]) is legal, but (([]))] and [(())[] are not legal.
   Question 1b: what if the sequence contains only '(' and ')'? Suggest a simpler solution.

2. **Monotone Stack.** A monotone stack is a stack whose elements are monotonically increasing or monotonically decreasing. Here we are concerned with the former. The basic operations in a monotone stack are the same as those in a stack: *push*, *pop*, and *top*. However, for every *push* operation, you need to pop out as few elements from the top as you can so as to maintain the monotonicity. For example, suppose the current stack is <1, 2, 6, 7, 8>, and you are to add in a 6. You need to pop out 8, 7, and 6 from the stack, before you push in 6. And the result will be <1, 2, 6>.
   Questions:
   1) Suppose the stack is initially empty, and you are to add in N elements. What is the total number of push operations? What is the maximum possible total number of pop operations?
   2) Now we invent another operation: *query k*, where *k* is a variable which might change for each query. This operation should return the minimum value among the k most recently inserted elements (including the ones that have been popped out). Please implement the *query k* operation.

   A similar question can be found here: http://www.acerwei.com/downloads/MonotoneStack.pdf

3. Implementing Stack and/or Stacks

In this question, we will implement several kinds of Stacks depending on our need.

a) *Second Minimum Stack*: Second minimum Stack is a Stack implementation that would allow the following operations to be performed:

```
push n         | put the number n at the top of the Stack
pop            | remove and return the top of the Stack
peek           | return (but not remove) the top of the Stack
second_min     | return the second minimum value in the Stack
```

However, there are simple limitations: *1) all of the operations should be done in a constant number of operations that does not depend on the size of the stack at any point in time* and *2) you may only use two normal Stacks (at least one for the storing the value Stack) and one more integer*. You may assume that the inputs are unique (i.e. there are no repeating numbers).

To elaborate, consider the following sequence of operations separated by semi-colon: `push 5; push 6; push 7; push 3; push 4; second_min; pop; second_min;`. The answer for the first query is `4`, while the second query is `5`, and both should be done without looping through the Stack.

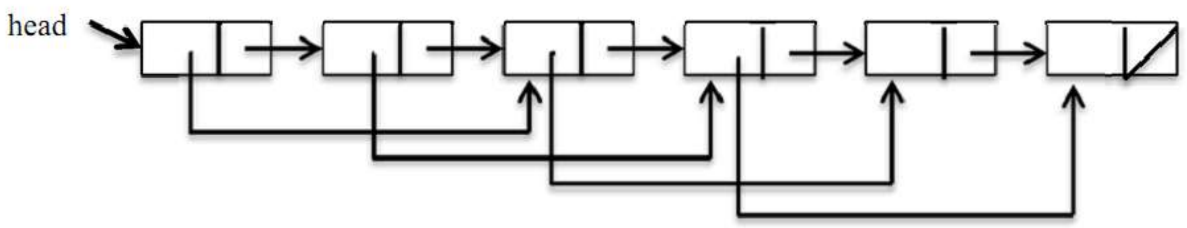
```
class SecondMinStack {
  private:
    /* Your Data Members... */
  public:
    SecondMinStack();
    void push(int val);
    int pop();
    int peek();
    int secondMin();
};
```

b) *Two Stacks*: How would you create two Stacks using only a single array? You can push into either one of the Stacks, however, both must utilize the entire array. There should be no case where one of the Stacks is full while the other still has some empty space despite the fact that both Stacks might have different sizes.

```
class TwoStacks {
  private:
    /* Your Data Members... */
    int _stacks[100];   // Example: you may choose other size
    // Add other data members if necessary, except array
  public:
    TwoStacks();
    void push1(int val); // Push to Stack 1
    void push2(int val); // Push to Stack 2
    void pop1();
    void pop2();
    void peek1();
    void peek2();
    void isEmpty1();
    void isEmpty2();
    void isFull();       // If Stack 1 is full, Stack 2 will be full
};
```

CS1020E: Data Structures and Algorithms I

4. [Jumping Linked List] In this question, you are asked to create a variant of LinkedList called Jumping Linked List. Each node in Jumping Linked List contains an extra pointer called kNext which points to a node k steps down the list. Take k = 2 for example.



a) Given that description and the Class definition below, implement the LinkedList together with the function to insert into the LinkedList while preserving the above property.

```
Class KListNode {
        Private: int _item;
                int k;
                KListNode* _kNext;
                KListNode* _next;
        Public: int getItem() {return _item;}
                KListNode* getKNext() {return _kNext;}
                KListNode* getNext() {return _next;}
};
```

b) Given the following scenarios, give your answer in terms of k and size of LinkedList.
1) You add a new element in back of the LinkedList. How many pointers have to be changed?
2) You remove an element from the middle of LinkedList of sufficiently large size. How many pointers have to be changed? Sufficiently large is several times the value of k.

A similar question can be found here: http://www.acerwei.com/downloads/SkippingLinkedList.pdf